ABSTRACT

AES (Advanced Encryption Standard) is state of the art symmetric cipher. This study illustrates a new approach to low cost and efficient implementation of AES cipher on 8 bit microprocessors. This crypto processor implementation is recommended for low speed applications where cost, availability and ease of developing and manufacturing comes in. Most efficient possible implementation of Galois field multiplication and key expansion on 8 bit microprocessors is demonstrated. Decryption algorithm is modified for two way communication without any overhead. Two modes of operations are considered, Electronic Codebook and Counter mode. Counter mode is further optimized for lowest latency due to its pre-processing feature.

Key Words: TAES, Galois Field Multiplication, Key Expansion, Modified Decryption, Counter Mode, Pre-processing

INTRODUCTION

Confidential data is often sent on a physically unsecure channel. The easiest and simple way to make our data secure from eavesdropping is to encrypt it in such a way that it is almost impossible to steal the information during transmission. Several applications like password protected home or vehicle security, or smart card based authentication needs encryption either for authentication. Voice may needs encryption on land-line telephone, cellular phone, walkie talkie etc. All of these applications do not require high speed crypto processors, but the primary concern is the cost and vast availability of the processor core used to implement encryption. We cannot compromise on the level of security for cost, but we can compromise speed for cost as many applications do not require high speeds. Other than low latency, availability and ease of development and production are also the major concerns. Availability of hardware-based crypto processors is very limited and their production on a relatively small scale is also difficult.

Implementation of state of the art encryption algorithm on low cost and widely available microcontrollers provides a good and viable solution to the problem of cost, availability and ease of development and production. In this study, Advanced Encryption Standard (AES) is implemented on widely available and low cost processor cores, in many different ways for optimizing it for as low latency as possible. Decryption algorithm is modified to make encryption inverse of decryption, to ensure two way crypto communication without any overhead. AES key expansion is optimized so that keys for all rounds are pre-evaluated and occupies no RAM and also reduces time needed to expand the key. This study illustrates low cost implementation of AES, in both Electronic Codebook (ECB) and Counter (CTR) modes of operations. Electronic Codebook and Counter modes are considered. Counter mode is more emphasized and is optimized for more speed using extra memory, taking advantage of the pre-processing feature of counter mode.

The small microcontroller cores can give only few kbps of throughput which is enough for confidentiality in real time voice or text communication and authentication purposes. Main purpose of this study is to show how to get most optimized implementation of AES on microcontroller in terms of maximum attainable speed, cost, area, ease of development, and availability of the processor cores.

The main latency adding and computational power consuming part of AES is Galois field or more precisely GF(2^8) multiplication that constitutes “Mix Columns” block for encryption and “Inverse Mix Columns” block for decryption. These functions are implemented using different methods in order to minimize the consumption of memory or to minimize the latency. If code memory consumption is minimized, latency increases and if latency is minimized code memory consumption increases. This study builds up using Atmel’s AT89S52 (because of its very
low cost i.e. less than a dollar and vast availability) having 8Kbytes of code memory so we can go for low latency than to conserve code space memory. Using Dallas Semiconductor’s DS89C420, DS89C430, DS89C440 or DS89C450 can further lessen the latency several times. So throughput we can get from AT89S52 is 4.555 Kbps for encryption and 2.978 Kbps for decryption. And by using DS89C4x0 we can get a throughput of several times more as compared to that of AT89S52. Another important thing that affect latency is the method in which key is added to the state. Different methods for the AES key expansion have been tested and selected the one delivering the lowest latency without compromising the security. Two modes of operation are considered i.e. Electronic Codebook mode and Counter modes.

The paper is organized in different sections outlining related work, presentation of the different ways to implement GF ($2^8$) multiplication and its optimization for the lowest latency and AES implementation in different modes of operation. Experimental results, conclusion and future work are outlined in final section.

MATERIALS AND METHODS

Related Work

Since the announcement of AES, the emphasis of the researchers in symmetric cryptography is on its implementation in a most efficient way with respect to cost, area and speed. A lot of work has been done on implementing it on hardware processor cores for very high throughput usually several hundred Mbps even several Gbps. For example Hämäläinen et al. (2006) have a throughput of 121Mbps at maximum clock frequency of 153MHz consuming 3.1kgates using 0.13µm CMOS technology. Liberatori et al. (2007) cipher on Altera Flex 10K FPGA has throughput of 11Mbps at 25MHz consuming 286 clock cycles using 957 logic cells and 6528 memory bits. Fan et al. (2007) architecture has a throughput of 360 Mbps using 4 S-Boxes simultaneously and a throughput of 114 Mbps while using only 1 S-Box at a clock frequency under 182 MHz using the TSMC (Taiwan Semiconductor Manufacturing Company) 0.18µm CMOS standard cell library and 7k gates. Stefan Tillich et al. (2008) have proposed a small enhancement for 8-bit Advanced Virtual RISC (AVR) achieving the performance upgrade by a factor of 3.6 encryption and decrypting under 1,300 clock cycles while proposed extensions cost 1.1kgates.

AES IMPLEMENTATION

Galois Field Multiplication

The Galois field multiplication, more precisely GF ($2^8$), is used in AES encryption as Mix Columns transformation and in AES decryption as Inverse Mix Columns transformations. In this study, it is implemented in several different ways and the most efficient method in terms of lowest latency is selected for final implementation as a crypto processor. The optimization of AES Galois field multiplication for 8 bit processors is explained below:

AES Mix Columns

Implementation of AES mix columns transformation can be categorized according to its latency and code memory it requires. AES mix columns transformation in its design is much faster than inverse mix columns transformation but still latency vary too much depending on its various implementations.

First Method

The first method is to do Galois field multiplication on runtime to reduce the consumption of code memory but it results in increased latency. In this method we have designed a subroutine for GF ($2^8$) multiplication with X as shown in Fig. 1. As we have only required to calculate the Galois field multiplication with X+1, X and 1. We can perform the mix columns operation using only this single subroutine. For X+1 we use the subroutine to get the GF ($2^8$) product of the operand with X and XOR the result with the operand. This method has the highest latency and is preferred where only single microcontroller unit is used both as a crypto engine and for the application requiring encryption. It is most effective in terms of area and cost.

Second Method

The second way is to use a lookup table for GF($2^8$) multiplication with X, as shown in Fig.2, to reduce the latency due to the time required for calculating the Galois field product on run time. In this method we calculate the product with X+1 using one lookup from the table and then XORing the result with the actual operand. This reduces the latency at the cost of 256Bytes of code memory which is consumed by the lookup table. This method is an intermediate solution where we have to balance both code memory and latency.
Third Method

The third and the most efficient method in terms of latency is to use two lookup tables for GF($2^8$) multiplication with both X and X+1. In this method all we do is just lookup from a table and no computationally complex Galois field multiplication, hence, improving the latency but we have to pay it in the form of 512Bytes of code memory used by the two lookup tables. This approach is suitable for Atmel’s AT89S52 as it has 8Kbytes of code memory.

![Flowchart](image1)

**Fig. 1. GF Multiplication with X using Subroutine**

![Flowchart](image2)

**Fig. 2. GF Multiplication with X using Lookup Table**

AES Inverse Mix Columns

AES inverse mix columns is the most computational power consuming part of the AES algorithm. AES mix columns is optimized for higher speed compromising the high latency of inverse mix columns because encryption is used more frequently than the decryption as a lot of applications need only encryption. So the main concern is to optimize the AES inverse mix columns for the lowest latency.

First Method

The first method uses the subroutines for the calculation of Galois field product at run time. The Galois field product in inverse mix columns transformation is many times more time consuming as compared to mix columns transformation. In this method we write subroutine for Galois field product with X, as shown in Fig. 1.
Here we have to multiply state with X many times as Fig. 3 demonstrates. Then using this subroutine we calculate the Galois field product with

- $X^4+1$,
- $X^4+X+1$,
- $X^4+X^2+1$ and
- $X^4+X^2+X$

that are equivalent to

- $09H$,
- $0BH$,
- $0DH$ and
- $0EH$.

![Flowchart](image)

**Fig. 3. GF Multiplication with $X^4$ using Subroutine**

$X^4+1$ is calculated by applying Galois field multiplication with X four times to the operand and then XOR the result with the operand. Similarly we can we find $X^4+X+1$, $X^4+X^2+1$ and $X^4+X^2+X$. This method has the highest latency but it is very memory efficient.

**Second Method**

The second method is to use lookup table for X. This reduces the time the processor takes to find the Galois field product with X by just a single lookup from the table. In this method we calculate Galois field product with

- $X^4+1$,
- $X^4+X+1$,
- $X^4+X^2+1$ and
- $X^4+X^2+X$
just like in first method except that now we do lookup from the table, as illustrated by Fig. 4, instead of calling subroutine to find Galois field product with X. This reduces the latency at the cost of 256Bytes of memory used by the lookup table. This method is a viable solution in the embedded system where both memory and speed are critical.

![Fig. 4. GF Multiplication with X^c using Lookup Table](image)

**Third Method**

The third method is most efficient in terms of latency. In this method we use four lookup tables for Galois field product with reducing the time needed to do very time consuming computation for finding the Galois field product. Here we have the lowest latency but we have used 1Kbyte of code memory for the four lookup tables. This method is the most preferable if the speed is the primary concern. As we have 8Kbyte of memory in Atmel’s AT89S52, this method can be implemented if the code memory is used wisely.

- X^4+1,
- X^4+X+1,
- X^4+X^2+1 and
- X^4+X^2+X

**AES Key Expansion**

The addition of round key is another important issue that affects the latency of encryption and decryption. I have implemented different methods for AES key expansion and its addition to the state and optimize it for best performance and memory efficiency. The first method is to expand key at the start of the encryption and store it in RAM in a static array that should not be overwritten. This method uses most of the available RAM because Atmel’s AT89S52 has only 256Bytes of RAM and the expanded key is of 144Bytes. This method is very efficient in terms of latency and code memory as we have to expand key just once after the device is powered up but it is extremely inefficient in use of RAM i.e. data memory. The second method is to devise a subroutine that should extract the key for the present round and there is no need to store the whole 144bytes in data memory except two previous keys. This method adds too much latency as key is obtained using a lot of computations for each and every round and each and every block of data. This method is efficient in terms of code and data memory if low latency is not required.

The third and the most efficient method is to store the expanded key in code space memory and just add it to the state with no need to expand the key at any stage. This method uses 144Bytes of memory but has very low latency adding factor and hence is the best possible solution for round key addition as the memory consumed is small compared to efficiency in latency.

**Full Duplex Crypto Communication without overhead**

AES encryption and decryption are not symmetric. It means although AES decryption is the inverse of AES encryption but AES encryption is not the inverse of AES decryption. So, in order to establish two way crypto communication we have to write both encryption and decryption routines on both peers. There is a way that we can establish two way communication without any overhead.
Default AES decryption algorithm is shown in Fig. 5. AES Decryption modified for two way communication without any overhead is shown in Fig. 6. On the decryption side we interchange inverse shift rows subroutine with inverse substitute byte subroutine and interchange inverse mix columns and add round key sub routines with inverse mix columns applied to the round key. As we know that inverse mix columns is the most time consuming part of AES. If we apply mix columns to round key at run time it adds too much latency. The solution is to store all round keys with mix columns applied to all round keys except first and the last round key for AES decryption in code memory. So, we do not have to do complex Galois field multiplication at run time and hence reducing the latency and also adds the capability of full duplex crypto communication without any overhead to our system.
Like other block ciphers AES can be implemented in any of five block cipher modes of operation as defined by NIST (FIPS 81) and 800-38A (Special Publication for counter mode). The block cipher modes of operations are:

- Electronic Codebook (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)
- Counter (CTR)

This study only concentrates on the Electronic Codebook and Counter modes of operation.

**Electronic Codebook (ECB) Mode**

As in electronic codebook mode each plaintext block is independently using the same key. This mode of operation is used for secure transmission of single values (e.g. an encryption key or a password). Disadvantage of ECB mode is that the same 128-bit data block always produces the same ciphertext, making exploitation of regularities in message possible. This study illustrates the most optimized form of AES working in Electronic Codebook mode. Inverse mix columns block which is used in only Electronic Codebook and Cipher Block Chaining mode is optimized for lowest possible latency. Equivalent inverse cipher is also illustrated which makes full duplex communication possible. All these features are intended to use with AES working in ECB mode of operation.
**Counter Mode**

The main emphasis of this study is the counter mode AES implementation because it provides sufficient security at least as much as provided by other modes listed above and many additional features which other modes cannot. Some features of counter mode are [6]:

- Hardware efficiency
- Software efficiency
- Pre-processing
- Random access
- Provable security
- Simplicity

We can use these features of counter mode in our implementation to for higher speeds.

**Using Pre-processing to Increase Speed**

Pre-processing can be done in counter mode if we use some kind of memory with our processing core like RAM, NVRAM, EEPROM or Flash Memory. After pre-processing the encryption reduces only to XOR operation. But we have an upper limit on pre-processing due to limitation of memory. To solve this problem we can exploit the fact that there is time when processing core have no incoming data to encrypt as shown in Fig. 7. This time can be utilized to encrypt counter and places it in the memory replacing the used encrypted counts stored in memory. In this way we can get in worst case i.e. when processor is busy all the time, speed equal to normal counter mode speed. On the average speed is improved many folds.

![Fig. 7. Pre-processing](image_url)
RESULTS AND DISCUSSION

After all of the optimizations steps, we get the following results for the encryption and decryption units:

Table 1. AES C Implementation on 8051

<table>
<thead>
<tr>
<th>Property</th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Memory Required</td>
<td>6.99 KB</td>
<td>6.29 KB</td>
</tr>
<tr>
<td>Key Size</td>
<td>128 bit</td>
<td></td>
</tr>
</tbody>
</table>

**Processor Core: Atmel AT89S52**

| Throughput @ Clock Frequency of 12 MHz | 4.605 Kbps | 2.873 Kbps |
| Throughput @ Clock Frequency of 33 MHz | 11.560 Kbps | 7.901 Kbps |
| Size of Processor Core                | 44A – TQFP, 10 x 10 mm Body Size, 1.0 mm Body Thickness[5] |

**Processor Core: Dallas Semiconductor 89C430**

| Throughput @ Clock Frequency of 12 MHz | 8.691 Kbps | 5.452 Kbps |
| Throughput @ Clock Frequency of 33 MHz | 23.901 Kbps | 14.994 Kbps |
| Size of Processor Core                | 44A – TQFP, 10 x 10 mm Body Size, 1.0 mm Body Thickness |

From results it is clear that the implementation approach is suitable for many applications requiring low speeds while keeping the cost for encryption very low. It can be used both for authentication and confidentiality. Authentication generally does not require higher speeds; therefore the implementation is best suited for application requiring encryption to encrypt password or a pin code. It can even encrypt voice in GSM cellular telephone network which is 13 kbps (in full rate mode).

CONCLUSION AND RECOMMENDATIONS

AES can be implemented on a small low cost microcontroller with speed enough for applications like authentication and real time text and voice communication requiring encryption while keeping cost extremely low. The implementation can be extended to the larger key sizes of 192 bits and 256 bits supported by AES. The implementation can also be accompanied with a public key algorithm (Diffie Hellman, RSA, etc) for transfer of encryption key for the first time. Systems requiring unconditionally secure communication the implementation can be extended to implement “One Time Pad”.

REFERENCES


