On Efficient Implementation of the Advanced Encryption Standard (AES)

Homayoun Shahri
Tufon Consulting
32732 Ballena
Dana Point, CA 92629
(949) 388-7100
homayoun@tufon.com

Nader Salessi
Caveolan, Inc.
Irvine, CA
(949) 510-6327
n.salessi@caveolan.com

ABSTRACT
In this paper, we describe a fast and efficient approach to implementing the newly selected Advanced Encryption Standard (AES). We present algorithms for interleaved key-expansion, as well as for computing the S-Boxes.

Categories and Subject Descriptors
D.3.3 [**Security**]: AES Algorithm –*KeyExpansion, S-Boxes.*

General Terms
AES, Cryptography, NIST, S-Boxes, Affine Transform, Key Expansion.

Keywords
AES, Cryptography, NIST, S-Boxes, Affine Transform, Key Expansion.

1. INTRODUCTION
In 2000 NIST selected the Rijndael as the algorithm for the Advanced Encryption Standard (AES). This is algorithm is known as the Federal Information Processing Standard (FIPS), publication 197.

FIPS chose Rijndael based on the following facts: When considered together, Rijndael's combination of security, performance, efficiency, ease of implementation and flexibility make it an appropriate selection for the AES.

Specifically, Rijndael appears to be consistently a very good performer in both hardware and software across a wide range of computing environments regardless of its use in feedback or non-feedback modes. Its key setup time is excellent, and its key agility is good. Rijndael's very low memory requirements make it very well suited for restricted-space environments, in which it also demonstrates excellent performance. Rijndael's operations are among the easiest to defend against power and timing attacks.

Additionally, it appears that some defense can be provided against such attacks without significantly impacting Rijndael's performance. Rijndael is designed with some flexibility in terms of block and key sizes, and the algorithm can accommodate alterations in the number of rounds, although these features would require further study and are not being considered at this time. Finally, Rijndael's internal round structure appears to have good potential to benefit from instruction-level parallelism.

With the realization that all communication must be secure, the apparent strength of Rijndael against various attacks, many firms have started designing and incorporating the AES into their devices. It is thus of utmost importance to implement AES as efficiently as possible. The implementation must be efficient in terms of gates to allow incorporation into low power devices, and efficient in terms of speed to allow fast throughput.

While we can increase the throughput and reduce the number of gates (area), there is the issue of latency. In AES, sub-keys are generated outside the main loop (rounds). The key-expansion thus adds to the latency. Some studies suggest that the vast majority of the internet packets are 64-byte long. In such cases the penalty paid due to latency is severe. We will also address the issue of latency and suggest a minor modification in the algorithm that enables interleaving the key-expansion with the AES rounds.

In pseudo code the AES algorithm is described as follows:

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
        byte state[4,Nb]
        state = in
        AddRoundKey(state, w[0, Nb-1])
        for round = 1 step 1 to Nr–1
                SubBytes(state)
                ShiftRows(state)
                MixColumns(state)
                AddRoundKey(state, w[round*Nb,
        (round+1)*Nb-1])
        end for
        SubBytes(state)
        ShiftRows(state)
        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
        out = state
end
```


ISPC Conference '03, March 31-April 3, 2003, Dallas, TX.

As can be seen in order to speed up the rounds we need to run all the step of the rounds in parallel, since AES cipher operates on vectors of 128 bits.

2. S-Boxes

The S-Box in AES is a table of 256 bytes. It substitutes an output byte for in input byte. The most common method of implementing the AES S-Box is to store the S-Box and its inverse, resulting in two tables of 256 bytes each. An AES S-Box is constructed by:

1. Take the multiplicative inverse in the finite field GF(2^8), the element {00} is mapped to itself.
2. Apply the following affine transformation over GF(2)

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

for $0 \le i < 8$, where b_i is the ith bit of the byte, and c_i is the ith bit of a byte c with the value {63} or {01100011}.

Some designers have opted to decouple the Galois Field inversion from the affine transformation. They thus maintain one table with inverses in the field of GF(2^8), and apply the affine transformation to implement the S-Box. The inverse of the S-Box is implemented by applying the inverse of the affine transformation and then looking up a value in the same inverse table used in the implementation of the S-Box. Using this approach we can reduce the tables by a factor of 2.

The field GF(2^8) used in AES is constructed by the polynomial:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

The multiplications are carried out by multiplying polynomials of degree 7 module *m(x)*. The inverse table is constructed by noting that *a*={*3*}, or {*x+1*} in polynomial form is primitive in this field, then all the elements can be constructed from this by forming consecutive powers of this element.
However, we can also compute the values of the table to form the inverses using combinational logic. The computation is done in composite fields.

This algorithm reduces the problem of inversion in $GF(2^8)$ to the problem of division in the field of GF(16^2). Multiplications and inversions can be directly implemented in the field GF(16). The algorithm start by applying a linear transformation on an element of $GF(2^8)$, represented as binary vector. This linear transformation is a mapping from the $GF(2^8)$ onto $GF(16^2)$. Since these fields are isomorphic, this transformation and its inverse always exist. Let us first assume that such transform exists, and present the division algorithm, we will then present the mapping from $GF(2^8)$ onto $GF(16^2)$.

Let the element $A = a_0 + a_1x$ be a canonical representation of an element in $GF(16^2)$ generated by $g(x) = x^2 + x + q$, where q is an element of *GF(16)*, and let the representation of the inverse of A be the element B in $GF(16^2)$ be $B = b_0 + b_1x$. We are trying to compute $B=1/A$. It is fairly straight forward to compute b_0 and b_1 by noting that $A.B=1 \bmod g(x)$, and solving for the two unknowns. first computing the inverse of B, and then multiplying by A. If we carry out the necessary computations we can show that:

$$b_0 = (a_0 + a_1) / D$$

and

$$b_1 = a_1 / D$$

where

$$D = a_0(a_0 + a_1) + qa_1^2$$

Note that the multiplies, squares and divisions are in the field of GF(16), which are considerably easier to compute.

Now let us present the mapping from $GF(2^8)$ onto $GF(16^2)$. We illustrate this with example for the field $GF(2^8)$ generate by the polynomial:

$$m(x) = x^8 + x^4 + x^3 + x^2 + 1$$

Our goal is to determine a binary matrix T of size *(k x k)* which can be used to map the elements from $GF(2^8)$ onto $GF(16^2)$. As discussed earlier this transformation is invertible and is represented by T^{-1}, which does the mapping in reverse direction. It is clear that a primitive element of $GF(2^8)$ generated by a polynomial $r(x)=x^8+x^4+x^3+x^2+x+1$ must be mapped into a primitive element of $GF(16^2)$ generated by a polynomial $p(x)= x^2+x+\omega^{14}$. And *GF(16)* is generated by $q(x)= x^4+x+1$.
However, this is not sufficient, since we don't know which primitive element it should map to. In order to insure that the mapping is isomorphic, we must make certain that a primitive element of $GF(2^8)$ which is a zero of $r(x)$ is mapped to an element which is a zero of $p(x)$, the generator of $GF(16^2)$ and vice versa. Let A = $\{\alpha^7, \alpha^5, \alpha^5, \alpha^4, \alpha^3, \alpha^2, \alpha, 1\}$ be the standard basis for $GF(2^8)$, α is a primitive root. Each element of this field can be represented as a binary 8-vector, which the result of a linear combination of these vectors. We are looking for the 8 basis elements represented with respect to $GF(16^2)$ onto which the 8 basis elements of set A are mapped. It is clear that the element '1' is mapped to the element '1', element α is mapped to a primitive element β which is a zero of $p(x)$, and α^2 is mapped to β^2 and so on. We thus have:

$$T\alpha^i = \beta^i, i = 0,1,...,7$$

If we followed the above procedure we find the transformation matrix T:

$$T = \begin{matrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{matrix}$$

The columns of the matrix T represent the basis vectors for the aforementioned transformation. For example one recognizes the element '1' as the right most column, and the second column from right is the element β^{37} which is also a zero of $r(x)$, under the rules of operations in $GF(16^2)$. After the transformation is performed the element of $GF(16^2)$ represented as a two-tuple over $GF(16)$ are read as the lower and upper nibble of the resultant transformation.
In order to get the transformation matrix for the field generated by:

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

which is used in AES, we can use the above technique to convert from the $GF(2^8)$ generated by the above to the field $GF(2^8)$ generated by

$$m(x) = x^8 + x^4 + x^3 + x^2 + 1$$

The matrix T that we are looking for is then the combination (product) of the two matrices.

The S-Boxes can then be created by combining the affine transform with the Galois field inverter as discussed above. Note that the affine transform matrix can be combined with inverse of matrix T derived above. The inverse is very similar. In this case the inverse affine transform matrix is combined with matrix T derived above.

3. Key-Expansion

We now present a slight modification of the AES algorithm which allows for interleaving of the key-expansion with the AES rounds. The AES key-expansion for the 128-bit key size in pseudo code is:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
        word temp
        i = 0
        while (i < Nk)
                w[i] = word(key[4*i], key[4*i+1],
                key[4*i+2], key[4*i+3])
                i = i+1
        end while
        i = Nk
        while (i < Nb * (Nr+1)]
                temp = w[i-1]
                temp = SubWord(RotWord(temp)) xor
                Rcon[i/Nk]
                w[i] = w[i-Nk] xor temp
                i = i + 1
        end while
end
```

This code can be modified to:

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
        word temp
        i = 0
        while (i < Nk)
                w[i] = word(key[4*i], key[4*i+1],
                key[4*i+2], key[4*i+3])
                i = i+1
        end while
        i = 1
        while (i <= Nr)
                j = Nb * i;
                w[j] = w[j - NK] xor SubWord(RotWord(
                w[j-1]))xor Rcon [(unsigned int) (j / NK)]
                j=j+1
                w[j] = w[j - NK] xor w[j-1]
                j=j+1
                w[j] = w[j - NK] xor w[j-1]
                j=j+1
                w[j] = w[j - NK] xor w[j-1]
        end while
end
```

As can be seen the main loop now iterates only Nr times, which is the same as the number of rounds in the AES cipher. Likewise the inverse can be derived. However, not that in inverse key-expansion, we start from the last generated key in the forward direction and go backwards.

4. REFERENCES

[1] NIST. Federal information processing standards publication 197: *Advanced encryption standard*, 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[2] Christof Paar, "*Some Remarks on Efficient Inversion in Finite Fields*", presented at IEEE International Symposium on Information Theory B.C. Canada, Sept. 1995.

[3] Christof Paar, "*Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields,*" Ph.D. Thesis, June 1994.

[4] Nicholas Weaver and John Wawrzynek, *"High Performance, compact AES Implementations in Xilinx FPGAs"*, U.C. Berkeley BRASS group.