Constant-Time AES

David Rufino

March 2, 2014

1 Introduction

It's known that a naive software implementation of the AES symmetric cipher [NIS01] is vulnerable to timing attacks [Ber05]. [Ham09] presents a constant-time implementation using SSSE3 instructions. This note simply expands on the details required to implement such a routine.

2 Background

Recall that the AES symmetric encryption algorithm works on a 16-byte (128-bit) block, with either a 128,192 or 256-bit key size. The bytes are considered to be elements of the finite field $\mathbb{F}_{2^8}$ of order 256, with respect to the polynomial basis associated to the canonical representation

$$\begin{aligned} \mathbb{F}_{2^8} &\simeq \mathbb{F}_2\left[X\right] / \left(m(X)\right) , \\ m(X) &= X^8 + X^4 + X^3 + X + 1 \,. \end{aligned} \tag{2.1}$$

For example

$$01101010b = 0\text{x}6A = X^6 + X^5 + X^3 + X$$

Addition is defined in the obvious way, and multiplication is defined modulo the polynomial $m(X)$. In this paper we make a distinction between the $\mathbb{F}_2$-vector space $\mathbb{F}_2^8$ and the field $\mathbb{F}_{2^8}$. In plain terms the former does not have a preferred definition of multiplication a-priori. It's only after choosing a basis for the field $\mathbb{F}_{2^8}$ that we can identify the two.
The "state" of the AES algorithm is conceptually a 4×4 matrix with elements in $\mathbb{F}_{2^8}$. The convention is that the elements are with respect to the canonical basis, and that it is stored as a 16-byte array in which the columns are contiguous. Each round of the encryption algorithm consists of the following transformations on the state

- **SubBytes**

  To each element of the state apply the transformation (in the canonical basis)

  $$z \to Az^{-1} + b$$

  where the elements A, b are defined in the original spec [NIS01] and inversion is understood to take place in the finite field discussed above.

- **ShiftRows**

  Rotate the rows of the state matrix

- **MixColumns**

  Pre-multiply the state matrix by the following

  $$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

where we identify a number with its binary expansion in the polynomial basis. For example

$$\begin{aligned} 5 &= 1 + X^2 \\ 10 &= X + X^3 \end{aligned}$$

NB. this step is omitted on the final round.

- **AddRoundKey**

  The key expansion (see [NIS01] for details) for this round is added (i.e. XOR-ed) with the state.

The first is non-linear and so typically implemented with lookup tables. It's discussed below that in order to avoid lookup tables it's easier to work in a different basis. For efficiency reasons then, we work consistently in the tower basis (defined below). This requires a number of changes

- The key expansion must be converted to the tower basis
- The input cipher text or plaintext must be converted at the start of the algorithm
- The output cipher text or plaintext must be converted to the polynomial basis at the end of the algorithm
- The lookup tables (for example, scalar multiplication) must be with respect to tower basis

3 PSHUFB Instruction

The PSHUFB (shuffle) instruction operations on two 128-bit (16-byte) registers as follows

```
unsigned char a[16],b[16],r[16];
for (i = 0; i < 16; ++i)
  r[i] = (b[i] & 0x80) ? 0 : a[b[i] & 0x0f];
```

As the choice of the register a is arbitrary, due to the presence of the mask on b this may be viewed as an arbitrary function

$$\mathbb{F}_2^4 \to \mathbb{F}_2^8$$

or alternatively as an appropriately restricted function

$$\mathbb{F}_2^4 \oplus \mathbb{F}_2^4 \to \mathbb{F}_2^8$$

where points with the most significant bit set are mapped to zero. Implementing a function

$$\mathbb{F}_2^8 \to \mathbb{F}_2^8$$

is in general difficult, but if it is linear (e.g. basis change) then two shuffle operations will suffice, as with the following pseudo-code

```
out1 = pshufb(lookup_lo, and(in,0x0f0f0f0f0f0f0f0f));
out2 = pshufb(lookup_hi, and(in,0xf0f0f0f0f0f0f0f0)>>4);
out = xor(out1,out2)
```

4 Inversion with Permutations

Inversion in $\mathbb{F}_{2^8}$ is not linear (in any basis), but [Ham09] shows it is possible to reduce this operation to a linear combination of such functions with domain $\mathbb{F}_2^4$ by considering an alternative ("tower") representation of $\mathbb{F}_{2^8}$

$$\begin{aligned} \mathbb{F}_{2^4} &\simeq \mathbb{F}_2[\zeta]/(\zeta^4 + \zeta^3 + \zeta^2 + \zeta + 1) \\ \mathbb{F}_{2^8} &\simeq \mathbb{F}_{2^4}[t]/(t^2 + t + \zeta)\,. \end{aligned}$$

Note in this representation every element $z \in \mathbb{F}_{2^8}$ may be represented uniquely as

$$z = xt + y\bar{t} \quad x, y \in \mathbb{F}_4$$

where $\bar{t} = t + 1$ is the conjugate of t. The addition law is clear and multiplication is given by

$$(x_1 t + y_1 \bar{t}) \cdot (x_2 t + y_2 \bar{t}) = x_1 x_2 t + y_1 y_2 \bar{t} + (x_1 + y_1)(x_2 + y_2)\zeta$$

The inverse of a general element is given by

$$(xt + y\bar{t})^{-1} = \frac{yt + x\bar{t}}{xy + (x^2 + y^2)\zeta}$$

Alternatively it is shown in [Ham09] that inversion in $\mathbb{F}_{2^8}$ may be expressed in terms of inversions over $\mathbb{F}_{2^4}$ with the following formula

$$\frac{1}{xt + y\bar{t}} = \frac{t + \zeta}{\frac{1}{1/y + 1/\zeta(x+y)} + x} + \frac{\bar{t} + \zeta}{\frac{1}{1/x + 1/\zeta(x+y)} + y} \tag{4.1}$$

provided one is careful about dividing by zero. The advantage of this formula is it does not require any explicit multiplications, which are more difficult to implement using shuffle instructions.
To implement the inversion formula (4.1) consider the four maps

$$\psi_i : \mathbb{F}_{2^4} \to \mathbb{F}_{2^8}$$

corresponding to the inversions

$$\begin{aligned} z &\to z^{-1} \\ z &\to (\zeta z)^{-1} \\ z &\to \frac{t + \zeta}{z} \\ z &\to \frac{\bar{t} + \zeta}{z} \end{aligned}$$

In practice these are implemented with respect to the basis $\{1, \zeta, \zeta^2, \zeta^3\}$ for $\mathbb{F}_{2^4}$ and the one already noted for $\mathbb{F}_{2^8}$. Further we ensure that $\psi_i(0) = 128$ so that $\psi_j(\psi_i(0)) = 0$ and(4.1) is correct in all cases. It's clear that together with addition (XOR) this is enough to implement equation (4.1).

5 Basis Change

This section discusses the conversion between canonical basis and the tower basis. The simplest way is to find a root of the AES polynomial (2.1) in the tower basis. Using this allows one to easily construct an isomorphism between the two representation. For concreteness consider the unique isomorphism generated by

$$X \to (\zeta + \zeta^3)\, t\,.$$

In practice we are interested in the basis change from the polynomial basis to the tower basis

$$\{t, \zeta t, \zeta^2 t, \zeta^3 t, \bar{t}, \zeta\bar{t}, \zeta^2\bar{t}, \zeta^3\bar{t}\}\,. \tag{5.1}$$

Call this $\hat{\phi} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$. Actually this function is a linear isomorphism, and so may be decomposed as follows

$$\begin{array}{rcl}\hat{\phi}(x) & = & \hat{\phi}_{\mathrm{lo}}(x_{\mathrm{lo}}) \oplus \hat{\phi}_{\mathrm{hi}}(x_{\mathrm{hi}}) \\ \hat{\phi}_{\mathrm{lo}}, \hat{\phi}_{\mathrm{hi}} & : & \mathbb{F}_2^4 \to \mathbb{F}_2^8 \,.\end{array}$$

The two functions are then in a form which may be implemented with the nstruction PSHUFB. For future reference we note the inverse of the basis change given above is generated by

$$\begin{array}{rcl} t & \to & X + X^5 + X^7 \\ \bar{t} & \to & 1 + X + X^5 + X^7 \\ \zeta & \to & X^4 + X^6 \,. \end{array}$$

It may be preferable to consistently work in the "tower" basis to avoid converting representations on each round. In this case the state must be converted to and from the tower basis at the start and the end of the algorithm, and they key expansion must also be converted.

6 Affine component of SubBytes

As the affine map is linear, it may be implemented with two shuffle instructions followed by an XOR. If it is adjacent to another linear map in the algorithm then clearly the two maps may be merged for efficiency reasons. For example in the inversion algorithm described above, the final two operations are shuffles, and the linear portion of the affine map may be incorporated directly. Note that must be careful when working in the tower basis.

7 Shift Rows

The ShiftRows transformation and its inverse can naturally be implemented with a single shuffle instruction.

8 Mix Columns

Recall that the mix columns operation corresponds to multiplication by a circulant matrix over $\mathbb{F}_{2^8}$

$$M := \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

It may be shown that the inverse operation is multiplication by the circulant matrix

$$M^{-1} := \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix}$$

Let P denote the elementary circulant matrix

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

This, and its powers, are easy to implement using the shuffle instruction. Recalling that $n \times n$ matrices form a $\mathbb{F}_{2^8}$-algebra (with elements acting point wise), then we see

$$\begin{array}{rcl} M & = & 2I + 3P + P^2 + P^3 \\ & = & (I + P)(2I + P) + P^3 \end{array}$$

and

$$\begin{aligned} M^{-1} &= (8+4+2)\,I + (8+2+1)\,P + (8+4+1)\,P^2 + (8+1)\,P^3 \\ &= \left(I+P^2\right)\left[9\cdot(I+P)+4\cdot I\right]+2\cdot(I+P)+I \end{aligned}$$

Calculation of M requires one doubling, three shuffles and three additions.
Calculation of M^{-1} requires three scalar multiplications, two shuffles and five additions.
Note in the canonical basis the scalar multiplications can be calculated as follows

$$X^k \sum_{i=0}^{7} a_i X^i = \sum_{i=0}^{3} a_i X^{i+k} + X^k \left(\sum_{i=4}^{7} a_i X^i \right) \qquad k \leq 4$$

The first term is a left shift and the final term may be calculated by a lookup table as with the following pseudocode.

```
mult1 = and(x,0x0f)<<k
mult2 = pshufb(lookup_k,and(x,0xf0)>>4)
mult  = xor(mult1,mult2)
```

Working in the tower basis requires two shuffle instructions.

References

[Ber05] Daniel J Bernstein. Cache-timing attacks on aes, 2005. `http://cr.yp.to/antiforgery/cachetiming-20050414.pdf`.

[Ham09] Mike Hamburg. Accelerating aes with vector permute instructions, 2009. `http://shiftleft.org/papers/vector_aes/vector_aes.pdf`.

[NIS01] NIST. Advanced encryption standard, 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.