Optimized Software Implementations of *E2*

Kazumaro Aoki[⋆] and Hiroki Ueda[⋆⋆]

NTT Laboratories[⋆⋆⋆]

Abstract. This paper describes some techniques for optimizing software implementations of *E2* on various platforms. It is relatively easy to implement a byte-oriented cipher such as *E2* on an 8-bit processor, but it is difficult to implement it efficiently on a 32-bit processor or a 64-bit processor. In particular, this paper shows some optimization techniques for SPN (Substitution-Permutation Network) on 32- or 64-bit processors. They are also applicable to other byte-oriented ciphers.

Keywords. ***E2***, SPN, optimization, 32-bit processor, 64-bit processor, inverse

1 Introduction

NTT submitted ***E2*** [KMA$^+$98, MAK$^+$98] as an AES candidate in response to the call issued by NIST in 1997 [U97]. ***E2*** is a byte-oriented[4] cipher, and was designed to be fast on 8-bit processors as well as 32-bit processors, which are current standards, and 64-bit processors, which are considered to be the next generation standard. Since ***E2*** is byte-oriented, it is not obvious how to implement ***E2*** efficiently on 32- or 64-bit processors.

This paper describes some techniques for optimizing software implementations of ***E2*** on such processors. Optimization techniques are introduced for each part of ***E2***. In particular, the optimization techniques for SPN (Substitution-Permutation Network) on a 32- or 64-bit processor are applicable to other byte-oriented ciphers as well.

We believe that these optimization techniques strengthen ***E2*** as a top AES candidate even when implemented on various platforms.

2 Specification of *E2*

Figure 1 shows the outline of the ***E2*** encryption process. ***E2*** has a 12-round Feistel structure with a preprocess, IT-Function, and a postprocess, FT-Function. The decryption process is the same as the encryption process except for the order of the subkeys. Figure 2 outlines F-Function. F-Function consists of S-Function, P-Function, and BRL-Function.

Refer to [N98a] for details of the specification and notations.

[⋆] Email: `maro@isl.ntt.co.jp`

[⋆⋆] Email: `ueda@isl.ntt.co.jp`

[⋆⋆⋆] 1-1 Hikarinooka, Yokosuka-shi, Kanagawa-ken, 239-0847 Japan

[4] In this paper *1 byte* is defined as 8 bits.

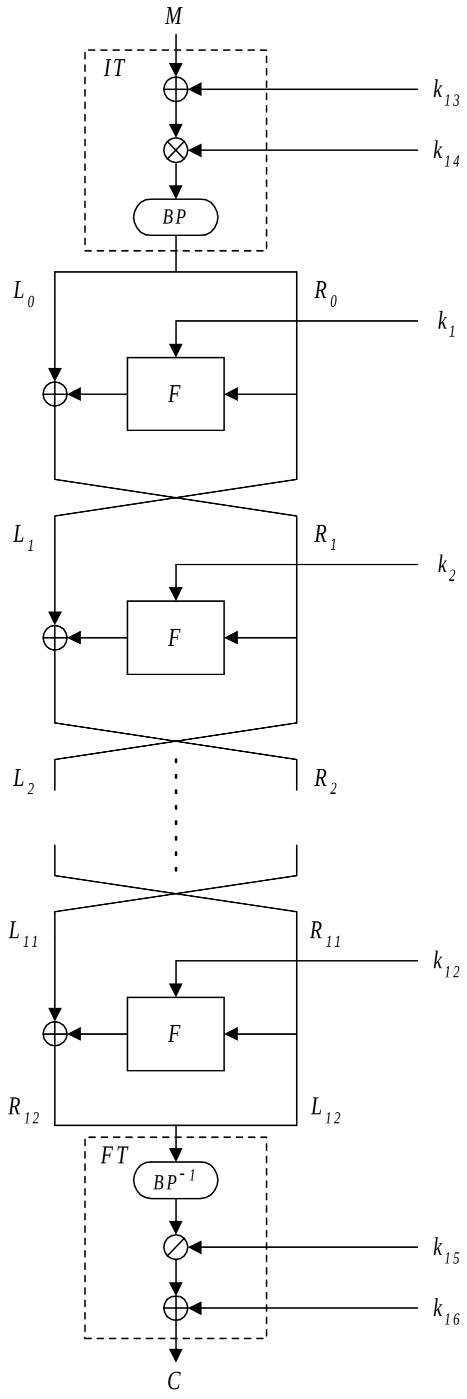

Fig. 1. Encryption process

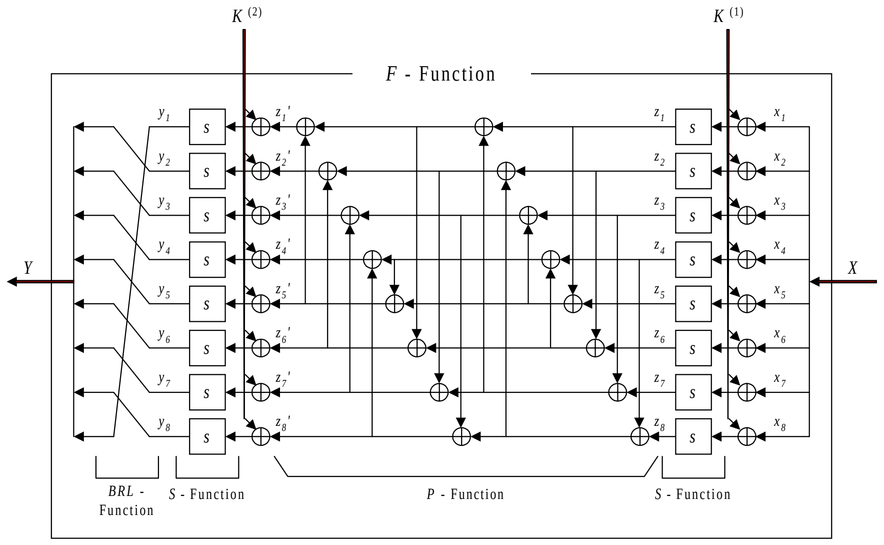

Fig. 2. F-Function

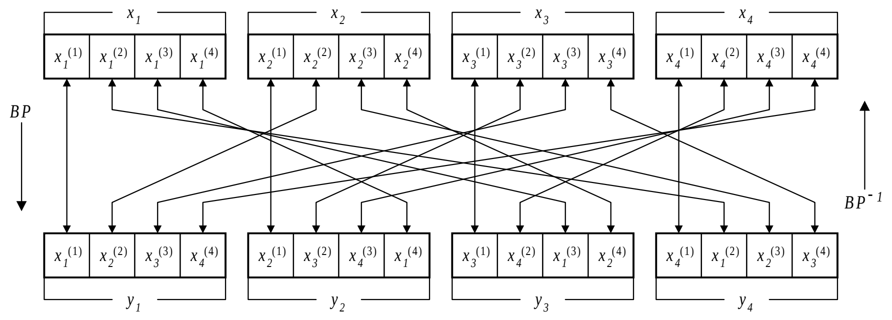

Fig. 3. BP-Function and BP^{-1}-Function

3 Optimization of each part of *E2*

Several optimization techniques were shown in [N98b]. However, this paper shows all known techniques including those described in [N98b].

3.1 Setup

3.1.1 $f(v_{-1})$

In the *E2* key scheduling part, G-Function shown in Fig. 4 is computed 9 times. In the first computation of G-Function, $f(v_{-1})$ can be calculated in the setup

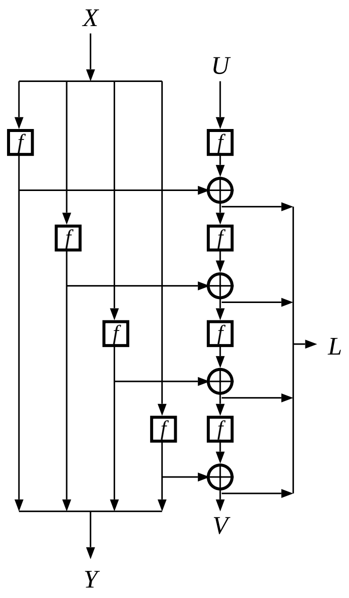

Fig. 4. *G*-Function

stage, since $U = v_{-1}$ holds and v_{-1} is a constant defined in the specification.

3.1.2 128- and 192-bit Key

When the key is 128- or 192-bits long, ***E2*** performs the same key scheduling tasks as in the case of the 256-bit key after padding the shorter keys with some constant values. Thus, f-Function which depends on only constants can be calculated in the setup stage. 18 or 9 f-Functions can be calculated for 128- or 192-bit keys, respectively, in the setup stage.

3.1.3 Inverse

The operation $\oslash$ in FT-Function requires an inverse in mod 2^{32}. This depends only on the key, i.e., it does not depend on plaintexts. Thus, the inverse can be calculated in the setup stage.

An inverse can be calculated by using the extended Euclidean algorithm. However, the extended binary GCD (ex., in [K97, Algorithm Y in p.646]) and Hensel lifting (ex., in [DK91, MODULAR-INVERSE algorithm in pp.235–236]) are more effective on a variety of platforms since the modulus has a special form.

Moreover, the Hensel lifting quadratic version proposed by Zassenhaus [Z69] is quite effective if the platform can use an effective 32-bit multiplier. We used Zassenhaus' algorithm to create Algorithm 1 for calculating inverses. It is useful for general processors whose word lengths are longer than 32 bits.

Algorithm 1. Calculation of $y = x^{-1} \bmod 2^{2^n}$. Let a and b be temporary variables, and $[z]$ be Gauss' symbol (which denotes maximum integer which does not exceed z), and the bit lengths of x, y, a, and b be 2^{2^n}.

```
Step 1: Input $x$. ($x$ is assumed as an odd integer.)
Step 2: Do the initial process as follows.
        1. $b := [\frac{x}{2}]$
        2. $a :=$ least significant bit of $b$
        3. $b := [\frac{ax+b}{2}]$
        4. $y :=$ least significant 2 bits of $x$
Step 3: Do the following for $i = 1, 2, \ldots, n-1$.
        1. $a := -by$
        2. $b := [\frac{b+ax}{2^{2^i}}]$
        3. $y := y + a \times 2^{2^i}$
Step 4: Output $y$.
```

3.2 Encryption Process

3.2.1 *S*-Function

S-Function in F-Function consists of 8 s-boxes whose input and output lengths are 8 bits. Figure 2 shows that 8 s-boxes can be calculated in parallel. Preparing the table $(x, y) \mapsto (s(x), s(y))$ halves the number of memory references. This technique requires as much as 64KB memory for the table, however, it is effective in the following cases.

1. The table can be stored in fast memory such as the 1st cache.
2. The 1st cache is hard to control such as in Java.

Referring to each s-box table is preferred if the size of the 1st cache is less than 64KB. Note that recent processors can cause a penalty when data that are not aligned on word boundary are accessed. For example, prepare table $x \mapsto (0,0,0,s(x))$ for a 32-bit processor instead of a simple 256 byte s-box table. Moreover, preparing the tables

$$\begin{aligned} x &\mapsto (0,0,0,s(x)) \\ x &\mapsto (0,0,s(x),0) \\ x &\mapsto (0,s(x),0,0) \\ x &\mapsto (s(x),0,0,0) \end{aligned} \tag{1}$$

eliminates the data position adjustment processes. However, we should ensure that the size of these tables does not exceed the size of the 1st cache.

3.2.2 BP-Function

BP-Function shown in Fig. 3 changes the order of bytes in IT-Function. A 32-bit or a 64-bit processor requires a large number of instructions if BP-Function is implemented in a straightforward manner since the number of instructions needed to handle byte operations is very large. Considering the processors requirements, we usually divide the input of F-Function into bytes for s-box input as described in Sect. 3.2.1. Thus, it is not necessary to follow the specification in terms of the byte order of an F-Function input, because no additional costs are incurred even if the byte-order is changed. When 16 bytes are divided into 2 eight bytes for input to the Feistel structure, we should efficiently extract 2 sets of 8 bytes which are outputs of BP-Function, which are left or right halves defined in the specification, and put them into registers. Since F-Function requires byte operations, the transformed F-Function which differs only in input byte order is not slower than the original F-Function.

To achieve this purpose, if we change the byte order

$$\texttt{01234567 89ABCDEF} \mapsto \texttt{05AF49E3 8D27C16B}$$

into

$$\texttt{01234567 89ABCDEF} \mapsto \texttt{09A345EF 812BCD67},$$

then the number of masking operations etc. is reduced to about a half. Note that each letter represents 1 byte.

To get the right ciphertext as defined in the specification, apply a similar technique to BP^{-1}-Function in FT-Function.

3.2.3 BRL-Function

BRL-Function is at the end of F-Function. If BRL-Function and S-Function are calculated at the same time, no time is required for BRL-Function. That is, we should put the output bytes from s-boxes into the right positions considering the effect of BRL-Function using (1), when bytes are changed to words.

3.2.4 P-Function

P-Function, which realizes linear transformation layer in F-Function, is represented as multiplication using an 8×8 matrix. If we consider the operation unit as a byte, the calculation requires 36 XORs, however, if we follow Fig. 2, only 16 XORs are required.

Algorithm 2 requires only 4 cycles if the algorithm is implemented on recent processors which offer pipelining, parallel execution, and 32-bit rotation. The byte order of the output does not match the specification, however, suitable coding may prevent a speed decrease, since each s-box is processed individually in S-Function.

Algorithm 2. Calculation of $Z' = P(Z)$. Let $\mathrm{RL}_b(X)$ mean b-byte left rotation of X.
Step 1: Input $(H, L) = ((z_1, z_2, z_3, z_4), (z_5, z_6, z_7, z_8))$.
Step 2: Do the operations as the following order.

cycle	Operation	order of H	order of L
1	$L := H \oplus L$	1234	5678
1	$H := \mathrm{RL}_2(H)$	3412	5678
2	$H := H \oplus L$	3412	5678
2	$L := \mathrm{RL}_3(L)$	3412	8567
3	$L := H \oplus L$	3412	8567
3	$H := \mathrm{RL}_1(H)$	4123	8567
4	$H := H \oplus L$	4123	8567

Step 3: Output $(H, L) = ((z'_4, z'_1, z'_2, z'_3), (z'_8, z'_5, z'_6, z'_7))$.

3.2.5 Substitution and Permutation

This section uses the notation

$$sb_1b_2\cdots b_n : x \mapsto (b_1 s(x), b_2 s(x), \ldots, b_n s(x)),$$

where $b_i \in \{0, 1\}$. For example, $s0010$ means $x \mapsto (0, 0, s(x), 0)$.

The substitution and the permutation in F-Function of ***E2*** is represented as

$$^T[z'_1 \ z'_2 \ \cdots \ z'_8] = P\ ^T[s(x'_1) \ s(x'_2) \ \cdots \ s(x'_8)]$$

using the matrix

$$P = \begin{bmatrix} 0\,1\,1\,1\,1\,1\,1\,0 \\ 1\,0\,1\,1\,0\,1\,1\,1 \\ 1\,1\,0\,1\,1\,0\,1\,1 \\ 1\,1\,1\,0\,1\,1\,0\,1 \\ 1\,1\,0\,1\,1\,1\,0\,0 \\ 1\,1\,1\,0\,0\,1\,1\,0 \\ 0\,1\,1\,1\,0\,0\,1\,1 \\ 1\,0\,1\,1\,1\,0\,0\,1 \end{bmatrix},$$

where x'_i is the XORed value of x_i and $K^{(1)}$ in Fig. 2, and the superscript T means matrix transposition.

[RDP+96] proposed an effective implementation of the substitution and permutation in SHARK. This section studies the implementation of substitution and permutation for 64- and 32-bit processors based on the implementation of SHARK.

64-bit processor. Using the implementation technique of SHARK directly means that tables

$$s01111101,\ s10111110,\ s11010111,\ s11101011,$$
$$s10111001,\ s11011100,\ s11100110,\ s01110011$$

are required. The computation cost of this is summarized as follows.

Required memory	16KB
Number of table references	8
Number of XORs	7

When the size of the 1st cache is less than 16KB, 8 tables described above may be generated from just $s11111111$ using masks. This case is summarized as follows.

Required memory	2KB
Number of table references	8
Number of masks	8
Number of XORs	7

32-bit processor. The previous section describing the implementation for 64-bit processors only discussed the implementation of $P(S(\cdot))$, since no effective implementation of $BRL(S(\cdot))$ has been found. This section considers the memory required for implementing the 2nd non-linear layer (substitution) in F-Function on a 32-bit processor, which causes good results.

Suppose that tables

$$s1000,\ s0100,\ s0010,\ s0001$$

as described in Sect. 3.2.1 are prepared for implementing the 2nd non-linear layer. They occupy a total of 4KB. Note that for time complexity we consider only $P(S(\cdot))$; we do not consider the 2nd non-linear layer.

Following the SHARK implementation technique directly, similarly to the case of 64-bit processors, tables

$$\begin{aligned} &s0111,\ s1011,\ s1101,\ s1110,\\ &s0011,\ s1001,\ s1100,\ s0110,\\ &s1000,\ s0100,\ s0010,\ s0001 \end{aligned}$$

are required for 32-bit processors. This case is summarized as follows.

Required memory	12KB
Number of table references	16
Number of XORs	14

If 4 bytes are stored in a 32-bit register in any order, the speed is the same, since the implemented process unit is a byte as described in Sect. 3.2.1. For example, changing the order of calculation as follows:

$$\begin{bmatrix} z'_1 \\ z'_8 \\ z'_5 \\ z'_4 \\ z'_7 \\ z'_2 \\ z'_3 \\ z'_6 \end{bmatrix} = \left[\begin{array}{cccc|cccc} 0&1&1&1&1&1&1&0 \\ 1&0&1&1&1&0&0&1 \\ 1&1&0&1&1&1&0&0 \\ 1&1&1&0&1&1&0&1 \\ \hline 0&1&1&1&0&0&1&1 \\ 1&0&1&1&0&1&1&1 \\ 1&1&0&1&1&0&1&1 \\ 1&1&1&0&0&1&1&0 \end{array}\right] \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \end{bmatrix},$$

means that tables

$$s0111,\ s1011,\ s1101,\ s1110,\ s0101,\ s1111,$$
$$s1000,\ s0100,\ s0010,\ s0001$$

are required, and memory references and XOR operations of high and low words corresponding to z_1, z_2, z_3, z_4 are the same. This improved case is summarized as follows.

Required memory	10KB
Number of table references	12
Number of XORs	11

Consider the case that required memory exceeds the size of the cache or the case that the latency[5] of memory references is problematic. For example, if we change the order of calculation to

$$\begin{bmatrix} z'_1 \\ z'_2 \\ z'_4 \\ z'_3 \\ z'_7 \\ z'_8 \\ z'_6 \\ z'_5 \end{bmatrix} = \left[\begin{array}{cccc|cccc} 0&1&1&1&1&1&1&0 \\ 1&0&1&1&0&1&1&1 \\ 1&1&1&0&1&1&0&1 \\ 1&1&0&1&1&0&1&1 \\ \hline 0&1&1&1&0&0&1&1 \\ 1&0&1&1&1&0&0&1 \\ 1&1&1&0&0&1&1&0 \\ 1&1&0&1&1&1&0&0 \end{array}\right] \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \end{bmatrix}$$

and prepare tables

$$s0111,\ s1011,\ s1101,\ s1110,$$
$$s1000,\ s0100,\ s0010,\ s0001,$$

memory references corresponding to z_5, z_6, z_7, z_8 are directly used for z'_1, z'_2, z'_4, z'_3, and z'_7, z'_8, z'_6, z'_5 are calculated as right 1 byte, right 2 bytes, left 1 byte, and left 2 bytes logical shifted from z'_1, z'_2, z'_4, z'_3, respectively. This case is summarized as follows.

Required memory	8KB
Number of table references	8
Number of XORs	11
Number of shifts	4

4 Implementation Results

We optimized ***E2*** implementations for several processors. Table 1 shows the results for the key scheduling part, and Table 2 shows the results for data randomizing part.

[5] Cycles after issuing an instruction before being able to access the result.

To achieve high performance on recent processors, it is important to consider instruction scheduling as well as decreasing the number of instructions. We achieved 2.32 [μops/cycle] parallel execution for a Pentium Pro and 1.73 [instructions/cycle] parallel execution for an Alpha processor (average values) using the implementations described in Table 2. We think that these implementations realize parallel execution efficiently.

Table 1. Key Scheduling Part

Processor	Key length (bits)	Speed (cycles/key)
Pentium Pro[a]	128	2076
	192	2291
	256	2484
H8/300[b]	128	14041
	192	15284
	256	16518

[a] IBM PC/AT compatible, Pentium Pro(200MHz), 64MB RAM, MS-Windows95, Borland C++ 5.02, ANSI C
[b] H8/300(5MHz) emulator on FreeBSD, assembly

Table 2. Data Randomizing Part

	Speed	
Processor	(cycles/block)	(bits/second)
Pentium Pro[a]	415	61.7M
Java VM[b]	2370	10.8M
Java VM[c]	28800	0.9M
Alpha[d]	587	130.8M
H8/300[e]	6374	100.5k

[a] IBM PC/AT compatible, Pentium Pro(200MHz), 64MB RAM, assembly
[b] IBM PC/AT compatible, Pentium Pro(200MHz), 64MB RAM, JDK 1.1.6 with JIT
[c] IBM PC/AT compatible, Pentium Pro(200MHz), 64MB RAM, JDK 1.1.6 without JIT
[d] Alpha AXP 21164A (600MHz), 8MB 3rd cache, 256MB RAM, Digital Unix 4.0, assembly
[e] H8/300(5MHz) emulator on FreeBSD, assembly

5 Conclusion

We analyzed each part of ***E2*** and studied how to implement them efficiently on various platforms. As a result, we achieved faster implementation on 32-bit processors, which are the current standard, and a 64-bit processor, which is considered to be the next generation standard, even though ***E2*** is a byte-oriented cipher.

NTT will continue to optimize ***E2*** implementation. The latest implementation results are available at `http://info.isl.ntt.co.jp/e2/`.

References

[DK91] S. R. Dussé and B. S. Kaliski Jr. A Cryptographic Library for the Motorola DSP56000. In I. B. Damgård, editor, *Advances in Cryptology — EUROCRYPT'90*, Volume 473 of *Lecture Notes in Computer Science*, pp. 230–244. Springer-Verlag, Berlin, Heidelberg, New York, 1991.

[K97] D. E. Knuth. *Seminumerical Algorithms*, Volume 2 of *The Art of Computer Programming*. Addison Wesley, third edition, 1997.

[KMA$^+$98] M. Kanda, S. Moriai, K. Aoki, H. Ueda, M. Ohkubo, Y. Takashima, K. Ohta, and T. Matsumoto. A New 128-bit Block Cipher *E2*. Technical Report ISEC98-12, The Institute of Electronics, Information and Communication Engineers, 1998. (in Japanese).

[MAK$^+$98] S. Moriai, K. Aoki, M. Kanda, Y. Takashima, and K. Ohta. S-box design considering the security against known attacks on block ciphers. Technical Report ISEC98-13, The Institute of Electronics, Information and Communication Engineers, 1998. (in Japanese).

[N98a] Nippon Telegraph and Telephone Corporation. *Specification of E2 — a 128-bit Block Cipher*, 1998. (`http://info.isl.ntt.co.jp/e2/`).

[N98b] Nippon Telegraph and Telephone Corporation. *Supporting Document on E2*, 1998. (`http://info.isl.ntt.co.jp/e2/`).

[RDP$^+$96] V. Rijmen, J. Daemen, B. Preneel, A. Bosselaers, and E. De Win. The Cipher SHARK. In D. Gollmann, editor, *Fast Software Encryption — Third International Workshop*, Volume 1039 of *Lecture Notes in Computer Science*, pp. 99–111. Springer-Verlag, Berlin, Heidelberg, New York, 1996.

[U97] U.S. Department of Commerce, National Institute of Standards and Technology. *Announcing Request for Candidate Algorithm Nominations for the Advanced Encryption Standard (AES)*, 1997. Federal Register: Volume 62, Number 177, pp.48051–48058, (`http://csrc.nist.gov/encryption/aes/aes-9709.htm`).

[Z69] H. Zassenhaus. On Hensel Factorization, I. *Journal of number theory*, Vol. 1, pp. 291–311, 1969.

This article was processed using the LaTeX macro package with LLNCS style