Fast Implementation of a 128bit AES Block Cipher Algorithm OCB Mode Using a High Performance DSP

Hyo-Won Kim, Su-Hyun Kim, Sun Kang, Taejoo Chang

Abstract—In this paper, the 128bit AES block cipher algorithm OCB (Offset Code Book) mode for privacy and authenticity of high speed packet data was efficiently designed in C language level and was optimized to support the required capacity of contents server using high performance DSP. It is known that OCB mode is about two times faster than CBC-MAC mode. As an experimental result, the encryption / decryption speed of the implemented block cipher was 308Mbps, 311Mbps respectively at 1GHz clock speed, which is 50% faster than a general design with 3.5% more memory usage.

Index Terms—Advanced Encryption Standard (AES), Offset Code Book (OCB), Digital Signal Processor (DSP), fast implementation, optimization.

1 INTRODUCTION

IN most crypto systems, many algorithms are applied for privacy and authenticity. It is required a lot of time and hardware resources to implement these algorithms. So an authenticated-encryption scheme is proposed for a shared-key encryption scheme whose goal is to provide both privacy and authenticity. This scheme can be just implemented to combine appropriately encryption algorithm and MAC (Message Authentication Code). Accordingly, this scheme is more efficient in performance and hardware resources than combining encryption algorithm and MAC. On a Pentium III processor, experiments by Lipmaa show that OCB is about 6.5% slower than the privacy-only mode CBC. The cost of OCB is about 54% of the cost of CBC encryption combined with the CBC MAC.

In this paper, the efficient method of optimizing a 128bit AES block cipher algorithm OCB mode using high performance DSP was proposed. A brief description of OCB mode is presented in chapter 2 and optimization methods that are used for improving performace and memory usage are explained in chapter 3. In chapter 4, experimental results that support the efficiency of the proposed design are provided and in chapter 4, there is a conclusion of this paper.

• *the Attached Institute of Electronics Telecommunications Research Institute, Yuseong P.O.Box 1, Daejeon 305-600, Korea. Tel: +82-42-870-2220, Fax: +82-42-870-2369 E-mail: hwkim@ensec.re.kr, hyuni76@ensec.re.kr, sunkang@ensec.re.kr, tchang@ensec.re.kr*

Manuscript received February 15, 2008; revised March 20, 2008.

2 OCB ENCRYPTION MODE

OCB mode was designed by Phillip Rogaway, who credits Mihir Bellare, John Black, and Ted Krovetz with assitance and comment on the designs. It is based on the authenticated encryption mode IAPM due to Charanjit S. Jutla. OCB combines the following features [1]:

- **Arbitrary-length messages + minimal-length ciphertexts:** Any string $M \in \{0,1\}^*$ can be encrypted; in particular, $|M|$ need not be a multiple of the block length n, which plaintexts are not padded to a multiple of n bits.

- **Nearly optimal number of block-cipher calls:** OCB uses $\lceil |M|/n \rceil + 2$ block-cipher invocations. Keeping low the number of block-cipher calls is especially important when messages are short.

- **Minimal requirements on nonces:** The entity that encrypts chooses a new nonce for every message with the only restriction that no nonce is used twice.

- **Efficient offset calculations:** In OCB mode, a sequence of offsets is generated in a particularly cheap way, each offset requiring just a few machine cycles.

- **Single underlying key:** The key used for OCB is a single block-cipher key, and all block-cipher invocations are keyed by this one key, saving space and key-setup time.

An encryption algorithm using OCB mode is as follows: Encryption under OCB mode requires an n-bit nonce, N. L is the encryption result of all zero input and $L(i) = L \cdot x^i$. The irreducible polynomial used in this paper is the following formula 1.

$$p_{128}(x) = x^{128} + x^7 + x^2 + x + 1 \quad (1)$$

$Z(i)$ denotes the offset that is very important in OCB mode. γ denotes n-bit canonical gray code and is generated using the following equation 2.

$$\begin{aligned} \gamma_i =& \gamma_{i-1} \oplus (0^{n-1}1 \ll ntz(i)), \\ & (\text{for } 1 \leq i \leq 2^n - 1) \end{aligned} \quad (2)$$

In equation 2, "$0^{n-1}1$" denotes the string of $n-1$ 0's and 1, respectively. $ntz(i)$ is the number of trailing 0-bits in the binary representation of i. With 2, the offset of OCB mode can be calculated by the following equation 3.

$$\begin{aligned} Z(i) &= \gamma_i \cdot L \oplus R \\ &= (\gamma_{i-1} \oplus (0^{n-1}1 \ll ntz(i))) \cdot L \oplus R \\ &= (\gamma_{i-1} \cdot L) \oplus (0^{n-1}1 \ll ntz(i)) \cdot L \oplus R \\ &= (\gamma_{i-1} \cdot L \oplus R) \oplus L(ntz(i)) \\ &= Z(i-1) \oplus L(ntz(i)) \end{aligned} \quad (3)$$

3 Design and Optimization in DSP

3.1 Design of AES OCB mode

Figure 1 illustrates OCB encryption. The message is $M = M[1]M[2]...M[m]$ and the noise is N. The resulting ciphertext is $C = C[1]C[2]...C[m]T$. The checksum is $M[1] \oplus ... \oplus M[m-1] \oplus C[m]0^* \oplus Y[m]$.

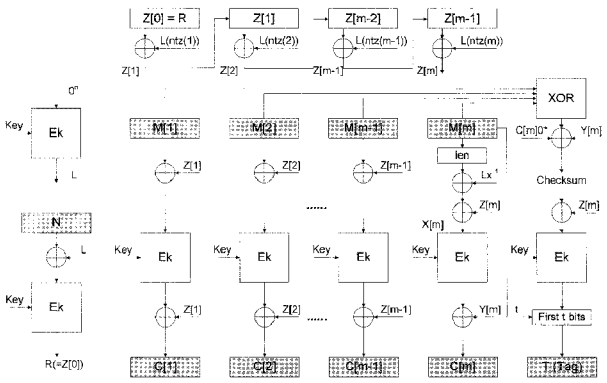

Fig. 1. Illustration of OCB encryption

```
typedef unsigned char block[16];

typedef struct _keystruct {
        unsigned rek[4*(AES_ROUNDS+1)];
        unsigned rdk[4*(AES_ROUNDS+1)];
        unsigned tag_len;
        block L[PRE_COMP_BLOCKS+1];
        block L_inv;
} keystruct;
```

```
keystruct *ocb_aes_init(void *enc_key, unsigned  tag_len,
        keystruct *key);
void ocb_aes_encrypt(keystruct *key, void *nonce,
        void *pt, unsigned pt_len, void *ct, void *tag);
int ocb_aes_decrypt(keystruct *key, void *nonce,
        void *ct, unsigned  ct_len, void *pt, void *tag);
```

Fig. 2. Definition of structure related key handle and designed algorithm function parameter

The designed algorithm is composed of three functions: **ocb_aes_init, ocb_aes_encrypt, ocb_aes_decrypt** [2]. Input and output parameters of each function and the definition of key related structure are illustrated in Figure 2.

3.2 Characteristics of DSP C64x

The TMS320C64x device is based on the second-generation high-performance, advanced VelociTI very-long-instruction-word (VLIW) architecture, making DSP suited for wireless infrastructure applications.

- Highest-Performance Fixed-Point DSPs
  -1.67-/1.39-/1.17-/1-ns Instruction Cycle
  -600-/720-/850-MHz, 1-GHz Clock Rate
  -Eight 32-Bit Instructions/Cycle

-Twenty-Eight Operations/Cycle
-4800, 5760, 6800, 8000 MIPS
- Eight Highly Independent Functional Units
-Six ALUs (32-/40-Bit), Each Supports Single 32-Bit, Dual 16-Bit, or Quad 8-Bit Arithmetic per clock cycle
-Two Multipliers Support Four 16 × 16-Bit Multiplies (32-Bit Results) per Clock Cycle or Eight 8 × 8-Bit Multiplies(16-Bit Results) per Clock Cycle
- Support Large Internal Memory (256K+8M Bit)

In this paper C64x is used in the following environment.

- Device: Texas Instrument TMS320VC6416
- Clock rate: 1GHz
- Memory: 16KB L1P Program Cache, 16KB L1D Data Cache, 1MB L2 Unified Mapped RAM/Cache
- Endian mode: Little endian

3.3 Optimization process in DSP

The optimization process is performed with the opened AES OCB source code [12] designed in C language using DSP c-compiler. The process is as follows

- After analyzing OCB mode architecture sufficiently, design efficiently in C language level at first.
- Add restrict qualifier to reduce potential pointer aliasing problems [9].
- Add loop iteration count to allow loops with indeterminate iteration counts to execute epilogs.
- Use memory bank pragmas and _nassert intrinsic to pass memory bank and alignment information to the compiler.
- Modify 8bit or 16bit arithmetic into 32bit arithmetic.
- Add the inline keyword to the callee's function definition. With the inline function expansion, most program excution speed is improved but program size is increased.

(1) Memory(Aliasing)

```
void rijndaelEncrypt(const u32 rk[], int Nr, const u8 * restrict pt,
        u8 * restrict ct)
{
    u32 s0, s1, s2, s3, t0, t1, t2, t3;
#ifndef FULL_UNROLL
    int r;
#endif /* FULL_UNROLL */

    s0 = GETU32(pt    ) ^ rk[0];
    s1 = GETU32(pt +  4) ^ rk[1];
    s2 = GETU32(pt +  8) ^ rk[2];

    ...
}
```

```
void rijndaelDecrypt(const u32 rk[], int Nr, const u8 * restrict ct,
        u8 * restrict pt)
{
    u32 s0, s1, s2, s3, t0, t1, t2, t3;
#ifndef FULL_UNROLL
    int r;
#endif /* FULL_UNROLL */

    s0 = GETU32(ct    ) ^ rk[0];
    s1 = GETU32(ct +  4) ^ rk[1];
    s2 = GETU32(ct +  8) ^ rk[2];

    ...
}
```

Fig. 3. Example of memory aliasing using restrict keyword

In Figure 3, optimization process is performed using a keyword "restrict". As shown in Figure 3 input/output parameters pt, ct are used regardless of earlier value in AES encrypt/decrypt function.

(2) Modify 16bit arithmetic into 32bit arithmetic

In OCB mode function, a block XOR arithmetic is used frequently, it is important to design block XOR arithmetic operation at once. Figure 4 is the example of modifiying 16bit XOR calculation into 32bit calculation.

(3) Inline function expansion

In general crypto systems, block cipher is not only used to operate encryption/decryption but also used to generate random numbers. From a developer point a view, it is not appropriate to design algorithms in a single file. So it is not easy to add inline keyword to

```
static void
xor_block(void *dst, void * restrict src1, void * restrict src2)
{
    ((u32 *)dst)[0] = ((u32 *)src1)[0] ^ ((u32 *)src2)[0];
    ((u32 *)dst)[1] = ((u32 *)src1)[1] ^ ((u32 *)src2)[1];
    ((u32 *)dst)[2] = ((u32 *)src1)[2] ^ ((u32 *)src2)[2];
    ((u32 *)dst)[3] = ((u32 *)src1)[3] ^ ((u32 *)src2)[3];
}

※ typedef unsigned int u32;
```

Fig. 4. Example of optimization modifying 16bit arithmetic into 32bit arithmetic

the callee's function definition. In this paper, inline function expansion is implemented with auto inline option(-oi) in c-compiler's build option. The size parameter is set 7274 refering to optimizer information size.

(4) Using precompiler directives

We use memory bank pragmas and _nassert intrinsic to pass memory bank and alignment information to the compiler. Figure 5 shows the example of optimization using precompiler directives.

```
static void shift_left(unsigned char *x)
{
    int i;

    ALIGNED_ARRAY(x);
    #pragma MUST_ITERATE(,,4);

    for (i = 0; i < 15; i++) {
        x[i] = (x[i] << 1) | (x[i+1] & 0x80 ? 1 : 0);
    }
    x[15] = (x[15] << 1);
}

static void shift_right(unsigned char *x)
{
    int i;

    ALIGNED_ARRAY(x);
    #pragma MUST_ITERATE(,,4);

    for (i = 15; i > 0; i--) {
        x[i] = (x[i] >> 1) | (x[i-1] & 1 ? 0x80u : 0);
    }
    x[0] = (x[0] >> 1);
}

※ #define ALIGNED_ARRAY(ptr) _nassert((int)ptr % 8 == 0)
```

Fig. 5. Example of optimization using precompiler directives

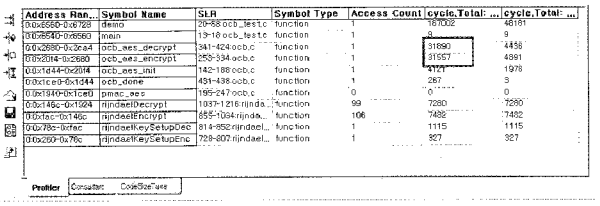

Fig. 6. Profiling result of the proposed implementation

4 Experimentation and Results

4.1 Profiling

Figure 6 shows the process of profiling to measure excution time of designed algorithm [11]. The size of input plain text used for simulation is 12,800 bits. As shown in Figure 6, the amount of cycles used for OCB mode encryption/decryption is 31,890 cycles, 31,557 cycles, which is 31.890 usec, 31.557 usec at TMS320c6416-1GHz.

4.2 Performance

Using the result of profiling, the performance of the proposed implementation is analyzed and compared with a general implementation (Release -o3 option).

(1) General implementation

- OCB Encryption: 48,115 cycles (12,800 bits) → 12,800 bits / (48,115 × 1ns) = 266.029 Mbps
- OCB Decryption: 47,427 cycles (12,800 bits) → 12,800 bits / (47,427 × 1ns) = 269.889 Mbps

(2) Proposed implementation

- OCB Encryption: 31,890 cycles (12,800 bits) → 12,800 bits / (31,890 × 1ns) = 401.380 Mbps
- OCB Decryption: 31,557 cycles (12,800 bits) → 12,800 bits / (31,557 × 1ns) = 405.615 Mbps

As shown above simulation results, the per-

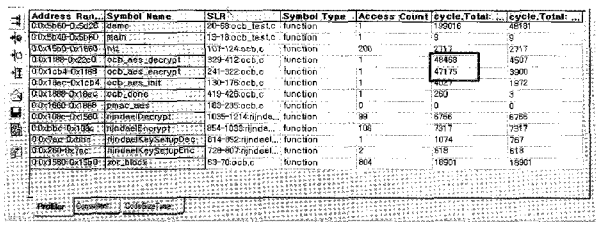

Fig. 7. Profiling result of a general implementation

TABLE 1
Comparison of Memory Usage

	General Implementation (Release -o3) (bytes)	Proposed Implementation (bytes)
ISRAM0	0x00002de0 (11,744)	0x000037e0 (14,304)
ISRAM1	0x0000fcc0 (64,704)	0x0000fcc0 (64,704)
Total	0x00012aa0 (76,448)	0x000134a0 (79,008)

formance of the proposed implementation is better than that of a general implementation by 135Mbps. This result shows 150% improvement of performance.

4.3 Memory Usage

A comparison between a general implementation (Release -o3) and the proposed implementation is illustrated in Table 1. As shown in Table 1, the proposed implementation uses 2,560 bytes more than a general implementation. It corresponds to 3.35% from the whole memory usage.

5 Conclusion

In this paper, the efficient method of optimizing a 128bit AES block cipher algorithm OCB mode using high performance DSP was proposed. Among authenticated-encryption schemes, it is known that OCB mode is about two times faster than CBC-MAC mode. The optimization process is performed with the opened AES OCB source code designed in C language using DSP c-compiler.

As a simulation result, the encryption / decryption performance of the implemented block cipher was 401Mbps, 406Mbps respectively at 1GHz clock speed. This result corresponds to 50% faster than general implementation with 3.5% more memory usage. The proposed optimization scheme is expected to be used in S/W crypto module designs for high performance requirement.

References

[1] Phillip Rogaway, *OCB: A block-cipher mode of operation for efficient authenticated encryption*, CCS'01, pp.196-205, November 2001.

[2] B. Choi, *Design of OCB-AES crypto processor VLSI*, KIMICS Journal, vol. 9, no. 8, pp. 1741-1748, June 2005.

[3] Y. Choi, *Real-time implementation of a tone sender/receiver on a high performance DSP*, ASK Journal, vol. 22, no. 4, pp. 276-285, March 2003.

[4] Y. Choi, *Implementation of a G.723 Annex A using a high performance DSP*, ASK Journal, vol. 21, no. 7, pp. 648-655, August 2002.

[5] J. Seo, *Real-time DSP implementation of IMT-2000 speech coding algorithm*, IEEK Journal, vol. 38, no. 3, pp. 304-315, May 2001.

[6] Y. Yang, *Real-time implementation of the 2.4kbps EHSX speech coder using a TMS320C6701 DSP core*, KICS Journal, vol. 29, no. 7C, pp. 962-970, September 2004.

[7] Texas Instruments, *TMS320C6416T fixed-point digital signal processors*, SPRS226H, August 2005.

[8] Texas Instruments, *TMS320C6000 optimizing compiler user's guide*, SPRU187L, May 2004.

[9] Texas Instruments, *TMS320C6000 programmer's guide*, SPRU198G, August 2002.

[10] Texas Instruments, *Code composer studio v3.1 IDE getting started guide*, SPRU509F, May 2005.

[11] Anand Krishnamurthy, Yiyan Tang, Cathy Xu, Yuke Wang, *An efficient implementation of multi-prime RSA on DSP processor*, ICASSP 2003, April 2003.

[12] http://www.cs.ucdavis.edu/~rogaway/ocb/ocb-code.htm

Hyo-Won Kim was born in Korea on February 17, 1977. He received the B.S. and M.S. degree in electronics from Yonsei University, Seoul, Korea in 1999 and 2001, respectively. He is currently working toward the Ph.D. degree in Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea. Since 2001, he has been the senior member of technical staff in the Attached Institute of Electronics and Telecommunication Research Institute (ETRI), Daejeon, Korea. His research interests include mobile communication, network security and system implementation.

Su-Hyun Kim was born in Korea on August 30, 1976. She received the B.S. and M.S. degree in computer engineering from Chonbuk National University, Korea in 1999 and 2001, respectively. She is currently working toward the Ph.D. degree in Korea University, Seoul Korea. Since 2001, she has been the senior member of technical staff in the Attached Institute of Electronics and Telecommunication Research Institute (ETRI), Daejeon, Korea. Her research interests include optical communication, network security and system implementation.

Sun Kang was born in Korea on January 15, 1969. She received the B.S. and M.S. degree in electronics from Chonbuk National University, Korea in 1991 and 1994, respectively. Korea in 1991 and 1994, respectively. From 1994 to 2000, she was a member and senior member of Technical Staff of Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. In 2000, she is joined the Attached Institute of Electronics and Telecommunication Research Institute (ETRI), Daejeon, Korea, where she is a senior engineering staff. She had been also with ETRI from 1994 to 2000. Her research interests include ASIC implementation, network security and system implementation.

Taejoo Chang was born in Korea on April 20, 1960. He received B.S. degree in Electrical Engineering from Ulsan University, Ulsan, Korea, in 1982, and M.S. and Ph.D. degrees in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1990 and 1998, respectively. From 1982 to 2000, he was a member and senior member of technical staff of Agency for Defense Development (ADD), Daejeon, Korea. In 2000, he joined the the Attached Institute of Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea, where he is now a principal member of technical staff. He became a Member (M) of IEEE in 2003, and a Senior Member (SM) in 2004. His research interests include design of cryptographic processors, network and RFID security, and statistical signal processing.