情報保護學會論文誌
第 21 卷 第 5 號, 2011. 10

모바일 환경을 위한 AES CTR Mode의 효율적 구현*

박 진 형,† 백 정 하, 이 동 훈‡
고려대학교 정보보호대학원

Efficient implementation of AES CTR Mode for a Mobile Environment*

Jin Hyung Park,† Jung Ha Paik, Dong Hoon Lee‡
Graduate School of Information Security, Korea University

요 약

인터넷 기술의 발달과 함께 스트리밍 서비스들이 많아지면서 이러한 서비스를 보호하기 위한 기술들이 개발되고 있다. 그 중 AES[1]의 CTR Mode는 OMA DRM, VoIP 그리고 IPTV 등의 스트리밍 서비스에서 안전한 정보 전송을 위해 쓰이는 암호화 기술로서, 전송되는 데이터의 암/복호화 병렬처리가 가능하다. 하지만 이러한 스트리밍 서비스를 사용하는 IPTV의 셋탑 박스나 모바일 디바이스는 제한된 연산 능력을 갖기 때문에, 이러한 환경을 고려하여 암호 알고리즘을 최적화하고 효율성을 높이는 것은 중요한 이슈가 된다. 따라서 본 논문에서는 AES-CTR Mode의 구현 로직을 개선하여 알고리즘 연산 속도를 개선하는 기법을 제안한다. 그리고 제한된 성능을 가지는 모바일 디바이스에서 제안한 기법을 구현하여 성능을 검증한다.

ABSTRACT

Recently, there are several technologies for protecting information in the lightweight device. One of them, the AES[1] algorithm and CTR mode, is used for numerous services(e.g. OMA DRM, VoIP, IPTV) as encryption technique for preserving confidentiality. Although it is possible that the AES algorithm CTR mode can parallel process transmitting data, IPTV Set-top Box or Mobile Device that uses these streaming service has limited computation-ability. So optimizing crypto algorithm and enhancing its efficiency for those environment have become an important issue. In this paper, we propose implementation method that can improve efficiency of the AES-CTR Mode by improving algorithm logics. Moreover, we prove the performance of our proposal on the mobile device which has limited capability.

Keywords: AES, CTR mode, Crypto Algorithm

I. 서 론

최근 네트워크 기술의 급속한 발전과 함께 인터넷 사용자 수가 급속히 증가하였다. 이에 따라 이러한 네트워크를 이용한 콘텐츠 전송 및 인터넷 전화(VoIP) 등 과 같은 인터넷 서비스들이 증가하고 있다. 이러한 인터넷 서비스들이 유·무선망을 통해 유료 서비스 및 개인 통화 정보 등을 전송할 경우, 그 내용이 제 3자에게 쉽게 노출되지 않도록 전송되는 정보의 기밀성을 유지하고 실시간 처리를 지원할 수 있는 정보보호 기술이 적용되어야 한다.

OMA(Open Mobile Alliance)의 DRM(Digital Right Management) V2.0 은 모바일 기기에서의 콘텐츠 보호를 위해 음악이나 동영상과 같은 연

접수일(2011년 1월 20일), 수정일(2011년 4월 20일), 게재확정일(2011년 5월 19일)
* 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단-미래기반기술개발사업(첨단융복합분야)의 지원을 받아 수행된 연구임(No. 2010-0020726).
† 주저자, ikarisj@naver.com
‡ 교신저자, donghlee@korea.ac.kr

속적인 미디어를 위한 PDCF 포맷에 암호화를 위한 AES CTR Mode를 포함하고 있다[2]. 또한 서비스의 특성상 고속의 암/복호화와 높은 안전성이 요구되는 IPTV 및 VoIP 서비스에는 실시간으로 전송되는 정보를 안전하게 보호하기 위해 Secure RTP(Real time Transport Protocol) 기술이 적용되고 있다[3]. Secure RTP는 보안이 고려되지 않은 기존의 RTP에서 정보전송의 안전성을 향상시킨 것으로 AES CTR Mode를 사용한다[4]. 이와 같이 AES CTR Mode는 OMA DRM, IPTV 그리고 VoIP 등의 스트리밍 서비스에 사용되며 정보의 기밀성 확보와 효율적인 암/복호화 처리를 지원한다. 하지만 IPTV의 셋탑 박스나 DRM 서비스 혹은 VoIP 서비스 등을 이용하는 모바일 디바이스들은 제한된 자원을 가지고 있으므로, 이러한 환경에 사용되는 암호화 알고리즘은 효율성을 고려하여 구현되어야 한다.

AES-CTR Mode는 블록의 수가 증가하는 동안, 입력으로 사용되는 IV의 최하위비트를 1씩 증가시켜 AES 암호 알고리즘에 입력하고 그 결과를 평문 블록과 XOR 연산을 수행하여 암호문 결과를 얻는 알고리즘이다. 이때, 매 블록마다 매우 적은 변화가 일어나는 CTR Mode의 특성과 최초 라운드의 정보 확산이 비교적 적은 AES 알고리즘의 성질로 인해 최초 블록의 첫 번째 라운드 결과는 그 다음 블록의 첫 번째 라운드 연산결과와 매우 적은 차이를 갖게 된다. 이러한 특성은 블록이 처리되는 동안 계속 이어지게 되고, 이러한 성질을 활용하여 알고리즘을 구현한다면 보다 효율적인 암호 처리 모듈을 구성할 수 있다.

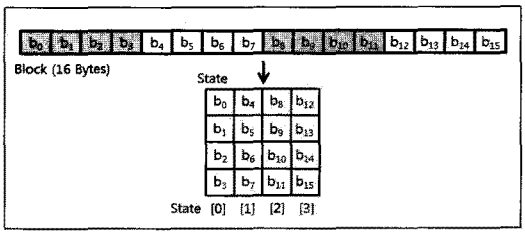
본 논문의 구성은 다음과 같다. 2장에서는 관련연구를 통해 AES 알고리즘의 기본적인 정보와 AES 알고리즘 최적화를 수행한 연구들을 살펴본다. 3장에서는 실시간 스트리밍 서비스를 사용하는 제한된 연산능력을 가진 모바일 디바이스에서 정보의 기밀성 유지를 위한 암호화가 효율적으로 작동할 수 있도록 내부 캐시를 이용한 AES CTR Mode (AES-CTR-C)의 구현을 제안한다. 4장에서는 구현된 결과물의 효율성을 분석하고, 5장에서 결론을 맺는다.

II. 관련 연구

2.1 AES 알고리즘 및 운영모드 개요

2.1.1 AES(Advanced Encryption Standard)

AES는 기존의 암호 표준인 DES의 안정성에 대한



[그림 1] AES Block-to-state Transformation

논란이 대두되자 미국 표준 기술 연구소(NIST)가 DES를 대체할 알고리즘으로 발표하여 미국 정보 표준으로 지정된 블록암호 알고리즘이다[1].

알고리즘에서 사용되는 블록 크기는 128 비트이며 128, 192, 256 비트의 키를 사용할 수 있다. 키의 크기에 따라 AES-128, AES-192, AES-256으로 구별되며, 각각 10, 12, 14 Round 로 동작한다.

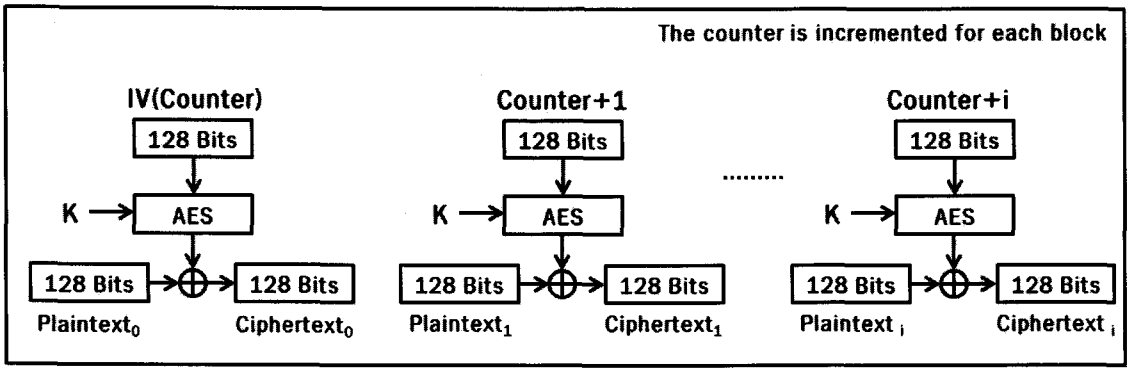
AES 알고리즘의 라운드는 4가지 변환 - *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey* - 으로 구성되어 있다. 각 변환은 *State*라고 불리는 4x4 행렬로 구성된 16 바이트 블록을 사용하여 동작하며, 각 라운드에 사용되는 라운드키는 S-box와 XOR, Rotation으로 구성된 일련의 프로세스(키 스케줄링)에 의해 비밀키로부터 생성된다.

AES의 데이터 블록은 128비트로 구성되며 16바이트를 원소로 하는 (1x16) 행렬로 표현할 수 있다. AES는 여러 횟수의 라운드를 사용하며, 각 라운드는 몇 가지 변환들로 구성된다. 데이터 블록은 각 단계에서 다른 단계로 이동함에 따라 데이터가 변형되는데, 각 단계 전후에 있는 데이터 블록을 스테이트(*State*)라고 정의한다. 데이터 블록과 마찬가지로 *State*는 16 바이트로 구성되며 바이트를 성분으로 하는 4x4 행렬로 나타낸다. 알고리즘의 시작 부분에서 데이터 블록의 바이트들은 [그림 1]과 같이 하나의 *State*에 열 단위로 삽입되고, 각 열에 대해서는 위에서 아래로 삽입된다.

*State*는 4열로 구성되어 있는데 왼쪽에서부터 열 단위로 *State*[0], *State*[1], *State*[2], *State*[3]으로 표현한다.

2.1.2 CTR Mode

CTR Mode[5][6]는 암호화/복호화가 동일한 구조이며, Counter를 사용함으로써 Key Stream이 의사난수성을 갖는다. [그림 2]에 나타난 것처럼 Counter라 불리는 입력 값이 암호 알고리즘에 의해



〔그림 2〕 CTR Mode Encryption

암호화된 결과가 평문과 XOR되어 암호문을 생성한다. n-bit의 카운터는 미리 정해진 값(IV)으로 초기화되고, 미리 정해진 규칙에 따라 값을 증가시킨다. 일련의 Counter 값들은 암호 알고리즘의 입력으로 동일한 값이 사용되면 안된다. 각각의 블록에 모두 다른 값이 사용되어야 하며, 동일한 키로 암호화되는 모든 메시지에 대해서 Counter 값은 반드시 서로 구별되어야 한다.

2.2 AES 알고리즘의 효율적 구현

AES 알고리즘의 구현을 보다 효율적으로 개선하고 구현하려는 시도는 하드웨어에 대한 효율성 개선과 소프트웨어 적인 로직 개선으로 분류 될 수 있다. *Rouvroy* 등은 Xilinx FPGA에서 Key schedule 부분과 Data path 부분을 합친 디자인을 제안하였다〔7〕. *Saqib* 등은 FPGA에서 Sequential 아키텍처와 Pipeline 아키텍처를 제안하였으며〔8〕, *Charot* 등은 Altera single-chip FPGA에서 single round를 모듈식으로 구현함으로써 Pipelining의 정도를 결정하는데 큰 유연성을 갖도록 하였다〔9〕. 이외에도 몇몇 효율적인 개선을 위한 시도가 이루어졌지만 주로 FPGA를 이용한 하드웨어 기반의 구현 방법이 대부분이었다. 이러한 하드웨어에 국한된 기법들은 응용 프로그램에 내장시켜 배포하는 것이 불가능하고 암호 모듈을 업데이트하기 어렵다는 단점이 있다.

이러한 하드웨어적 기법 연구와 함께 소프트웨어 관점에서 효율적으로 구현하기 위한 연구들도 수행되어 졌다. 정창호 등은 64-비트 프로세서인 Intel Core2 프로세서와 AMD Athlon64 프로세서에서 메모리 접근 명령어 비율을 낮추고 명령어의 개수를 최소화함으로써 AES 알고리즘을 고속으로 구현할 수 있는 기법을 제시하였다〔10〕. 하지만 이러한 특정 아키텍처에 기반한 최적화는 확장성을 크게 가지지 못하는 반면, AES 알고리즘 연산의 로직을 최적화 하고 자주 계산되는 부분의 Pre-Computation을 통한 효율성 확보 방법은 플랫폼에 의존적이지 않고 다양한 환경에 적용될 수 있다. 가장 널리 알려진 AES 알고리즘의 최적화 로직은 *Bertoni* 등이 제안한 32-비트 플랫폼에서 look-up 테이블을 이용한 AES 소프트웨어 구현 방법이다〔11〕. *Bertoni* 등은 이 논문에서 AES 알고리즘의 라운드 연산의 일부인 *SubBytes*, *ShiftRows*, *MixColumns*를 결합하고, 미리 계산된 look-up 테이블을 생성하여 라운드 연산에 적용하였다. 따라서 AES 알고리즘 자체는 256 Bytes의 S-Box 메모리를 필요로 하는 반면에 *Bertoni* 등의 기법은 256개의 Entry를 갖는 32 비트 테이블 4개가 필요하며, 따라서 총 4096 바이트의 메모리 공간이 요구된다. 또한 *Bernstein* 등은 AES 알고리즘 CTR Mode의 특성을 이용하여, 연산상의 일부 정보를 저장하여 재사용함으로써 알고리즘을 속도를 높일 수 있는 기법을 제안하였다〔12〕. CTR Mode의 특성상 16 바이트의 Counter 값 중 15 바이트는 256 블록을 처리하는 동안 동일한 값으로 유지되며 오직 한 바이트만 값이 변한다. *Bernstein*은 이러한 특성을 이용하여 초기 라운드에서 해당 바이트를 제외한 나머지 부분의 연산 결과를 저장하여 256 블록이 처리되는 동안 재사용하였다. Round 1에서도 4개의 4 바이트 입력 *State* 중 오직 하나의 *State*만이 매 블록마다 값이 변하는 특징이 있다. 따라서 초기 라운드에서와 마찬가지로 해당 *State*와 관련이 없는 연산 결과는 저장되어 256 블록을 처리하는 동안 재사용될 수 있다.

이러한 기법은 *Hongjun Wu*가 최초 제시하고 구

현한 기법으로서 Crypto++ 구현물[13], 혹은 *Hongjun Wu*의 AES 알고리즘 공개 코드[14]에서 확인할 수 있다.

III. 제안하는 방법 : AES-CTR-C

AES CTR Mode는 최초 블록에 입력된 IV를 기점으로 매 블록마다 LSB를 1씩 증가시켜 다음 블록의 입력으로 사용한다. 이때 AES 알고리즘의 첫 번째 연산은 *AddRoundKey*()로서 입력된 데이터 블록을 그대로 키와 XOR 연산에 적용한다. 단일블록의 길이가 16 바이트인 AES 알고리즘이기 때문에 LSB로부터 1씩 증가한 카운터가 나머지 전체 블록에 영향을 주기 위해서는 상당한 양의 데이터 프로세싱을 거쳐야 한다. 따라서 *Bernstein*이 소개한 *Hongjun Wu*의 방법[12]과 같이 알고리즘 내부적으로 매번 같은 입력 값을 *AddRoundKey*()에 적용하지 않고 계산된 결과를 재사용한다면 효율적인 결과를 도출할 수 있다. 또한 *AddRoundKey*() 이외에 첫 번째 라운드에 적용되는 *SubBytes*(), *ShiftRows*(), *MixColumns*()의 특성을 잘 활용한다면 추가적인 연산 결과의 재사용이 가능하다.

본 장에서는 다음과 같은 4가지의 AES-CTR Mode 구현 최적화 기법을 소개한다.

- AES-CTR-C_rd$_0$: AES Initial Round 에서 *AddRoundKey*() 연산의 결과를 캐시에 저장하고 재사용 하는 기법이다. *Hongjun Wu*가 15 바이트의 데이터를 저장하는 반면에, 본 제안에서는 12 바이트의 정보를 재사용함으로써 캐시에 저장된 정보의 잦은 갱신을 최소화 한다.
- AES-CTR-C_rd$_1$: AES Round 1에서 *SubBytes*(), *ShiftRows*(), *MixColumns*(), AddRoundKey()의 연산결과 중 12바이트 정보를 캐시하여 재사용하는 방법으로 *Hongjun Wu*에 의해 소개되었다.
- AES-CTR-C_rd$_2$: AES Round 2에서 MixColumns()연산 과정 중 매 블록마다 12 바이트 정보가 반복되는 점을 이용하여 해당 데이터를 캐시하고 재사용하는 방법을 새롭게 제안한다.
- AES-CTR-C_rd$_{1+}$: AES Round 1에서 AES-CTR-C_rd$_1$ 은 12바이트의 정보를 캐시에서 읽어들여 사용하고 4바이트 정보는 매 블록마다 새로 계산한다. 본 논문에서는 암호 인스턴스 초기화 단계에서 매번 계산되는 4바이트 정보의 Pre-Computation Table을 생성하여 효율성을 극대화 하는 기법을 제안한다.

3.1 Cache State of Initial Round (AES-CTR-C_rd$_0$)

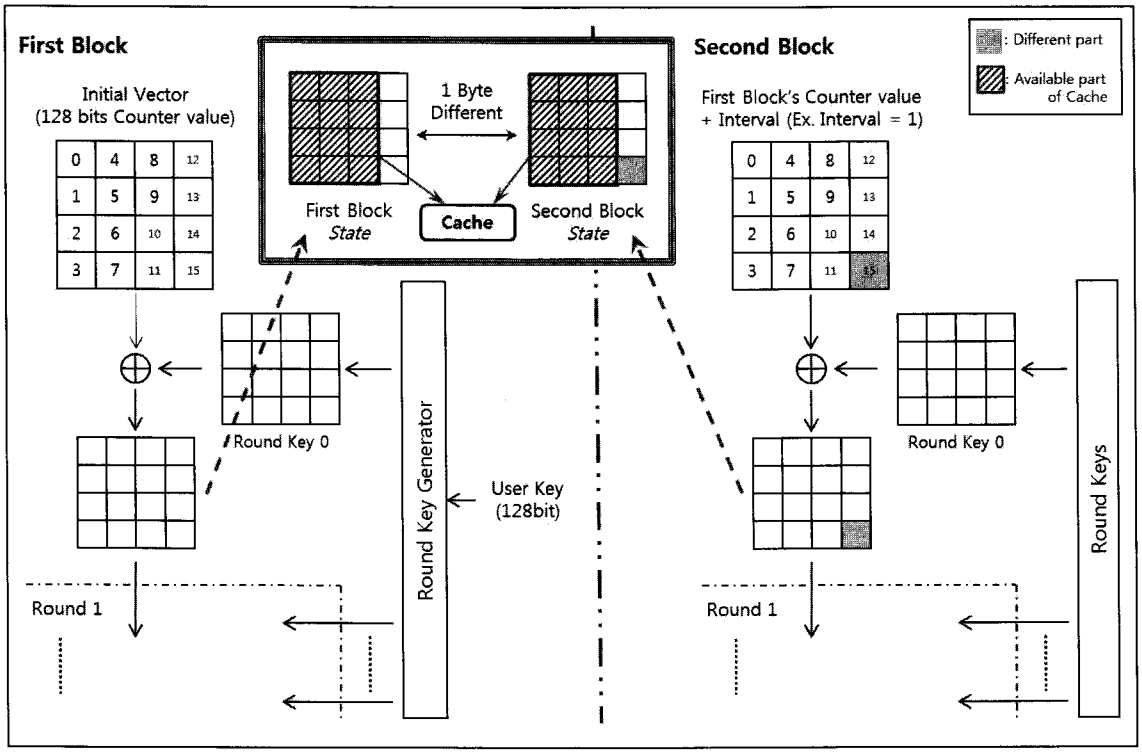
AES CTR Mode 알고리즘은 초기 라운드에서 암호 알고리즘의 입력 값과 Cipher key(Round Key)를 XOR한다. 첫 블록의 입력 값을 IV (Initialization Vector)로 설정하고, 미리 정해진 규칙에 따라 IV의 값을 증가시켜 다음 블록의 입력 값으로 사용한다. 첫 블록의 입력 값인 IV와 바로 다음 블록의 입력 값 Counter value는 Carry가 발생하지 않는다면, 증가한 값에 영향을 받는 마지막 1 바이트만 값이 다르고 나머지 값은 같다. 예를 들어, 값을 1씩 증가시키는 Counter를 사용하며, IV$_0$값을 1로 설정했다고 가정하면, IV$_0$값과 카운터에 의해 증가된 값 IV$_1$(다음 블록의 입력값)은 다음과 같이 마지막 1 바이트만 다른 값을 갖는다. IV를 { *State*[0], *State*[1], *State*[2], *State*[3] }이라 할 때,

IV$_0$: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01

IV$_1$: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x02

결국 입력 값과 첫 번째 Round Key가 XOR되는 초기 라운드에서 각각의 데이터 블록들은 같은 Round Key 값과 XOR되기 때문에 그 결과도 IV$_0$, IV$_1$과 마찬가지로 마지막 1 바이트만 다른 값을 갖는다.

*Hungjun Wu*는 이러한 특성에 대해서 초기 라운드의 결과에서 16 바이트 중 마지막 1 바이트를 제외한 나머지 15 바이트를 저장하여 256(2^8) 블록동안 재사용하는 방법[12]을 제시하였지만, 본 논문에서는 초기 라운드의 결과 16 바이트 중 마지막 4 바이트를 제외한 나머지 12 바이트를 저장함으로써 2^{32}-1 블록을 연산하는 동안 재사용할 수 있는 방법을 제안한다. 즉, 초기 라운드의 결과로 저장된 *State*에서 값이 변하지 않는 부분인 *State*[0], *State*[1], *State*[2] 부분을 캐시하여 다음 블록의 연산 시에 재사용한다. [그림 3]은 초기 라운드 과정과 결과 *State*의 캐시할



〔그림 3〕 CTR Mode에서 첫 블록과 두 번째 블록의 초기 라운드

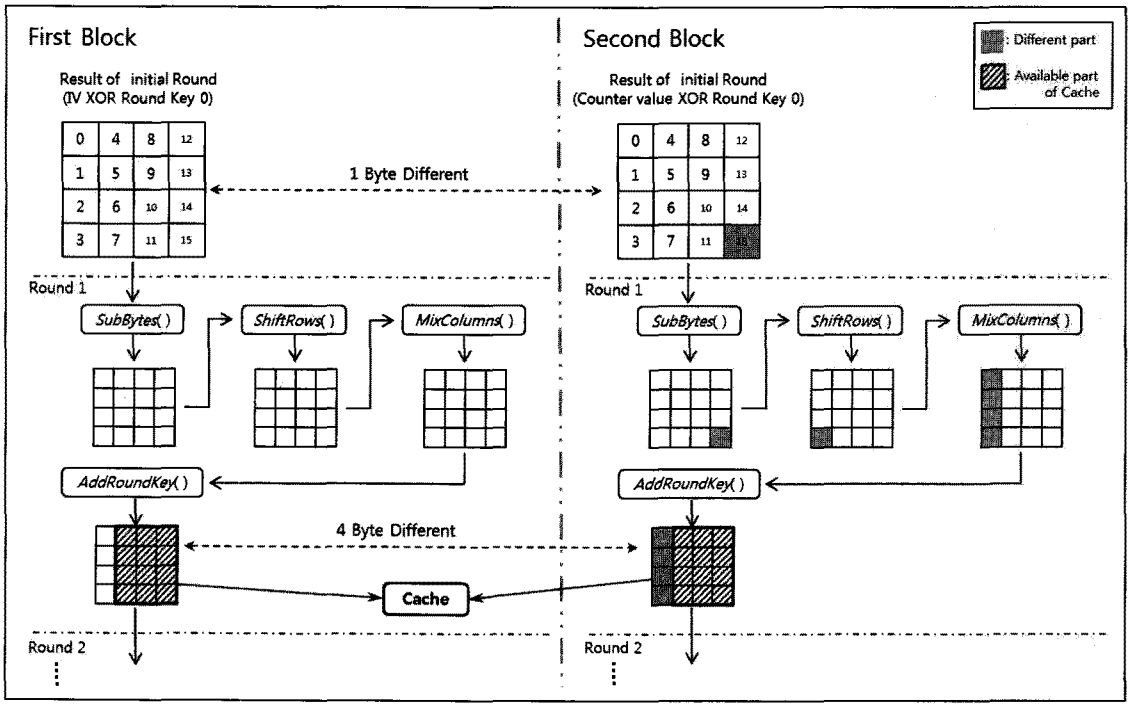
수 있는 부분을 나타낸 것이다.

Counter 값의 증가에 따라 Counter value 데이터 블록의 12번째 바이트 값이 변경되면서 Carry를 발생시켜 11번째 바이트에 영향을 미칠 때까지 캐시된 정보를 사용할 수 있다. 예를 들어, 카운터 증가 값이 1 이고, IV값이 0 인 경우 Counter value의 11번째 바이트에서 Carry가 발생하는 블록은 4,294,967,297번째 블록이 된다. 따라서 이 경우 4,294,967,295개의 블록을 계산하는 동안은 초기 라운드에서 *State*의 일부분에 대해 입력 값과 Round Key의 XOR 연산을 수행하는 대신 저장된 정보를 사용할 수 있다. AES 알고리즘의 한 블록은 16 바이트이므로, 위와 같은 경우 약 65.5 GBytes 마다 한 번씩만 캐시된 내부 데이터를 갱신해주면 된다.

3.2 Cache State of Round 1 (AES-CTR-C_rd$_1$)

초기 라운드를 거친 이후에 처리되는 데이터는 첫 번째 라운드에 입력으로 활용된다. 이때 첫 번째 라운드에서도 역시 데이터를 저장하여 다음 블록에 활용할 수 있다. 다만 초기 라운드보다 캐시된 데이터의 갱신이 자주 일어난다. 〔그림 4〕와 같이 Round 1의 입력이 되는 초기 라운드의 *State*는 첫 번째 블록과 두 번째 블록이 마지막 1 바이트만 서로 다른 값을 갖는다. 비록 Round 1 과정을 통해 *State*의 서로 다른 값을 갖는 마지막 1 바이트는 *ShiftRows*()와 *MixColumns*()의 단계에서 이동되고 확산되지만, 그 영향은 하나의 *State*에 국한된다. 따라서 〔그림 4〕와 같이 Round 1의 결과로 출력된 *State*에서 *State*[0]을 제외한 나머지 *State*들을 캐시하여 사용할 수 있다. 이 때 Counter 값의 증가에 따라 Counter value의 15번째 바이트에서 Carry가 발생하여 초기 라운드 *State*의 14번째 바이트 값이 바뀌면, 캐시 해두었던 *State* 값들을 업데이트 해주어야 한다. 이는 이미 *Hongjun Wu*에 의해 구현된 기법으로 *Hongjun Wu*의 AES 알고리즘 공개 코드 〔14〕에서 확인할 수 있다.

Carry로 인해 초기 라운드의 결과 *State*에서 14번째 바이트 값이 함께 바뀐 경우에도 Round 1 과정 후의 *State* 결과 값 중 *State*[2]와 *State*[3]은 여전히 같은 값이 유지된다. 이러한 경우를 고려하여 14



〔그림 4〕 첫 블록과 두 번째 블록의 Round 1 수행 결과

번째 바이트 값이 바뀐 경우 *State*[2], *State*[3]을 연산하지 않고 캐시된 데이터를 불러오도록 구현할 수도 있다. 하지만 이 경우, 연산의 횟수를 줄여서 얻을 수 있는 이득보다 Carry Check로 인한 오버헤드가 더 크기 때문에 전체 알고리즘의 수행 속도는 앞의 방법보다 느리다.

결국 효율성을 고려하였을 때 초기 라운드에서 캐시된 *State* 정보가 사용될 수 있는 최대 블록 수는 2^{32}-1 이며, Round 1에서 캐시된 *State* 정보가 사용될 수 있는 최대 블록 수는 2^{8}-1 이다.

3.3 Cache Data of Round 2 (AES-CTR-C_rd$_2$)

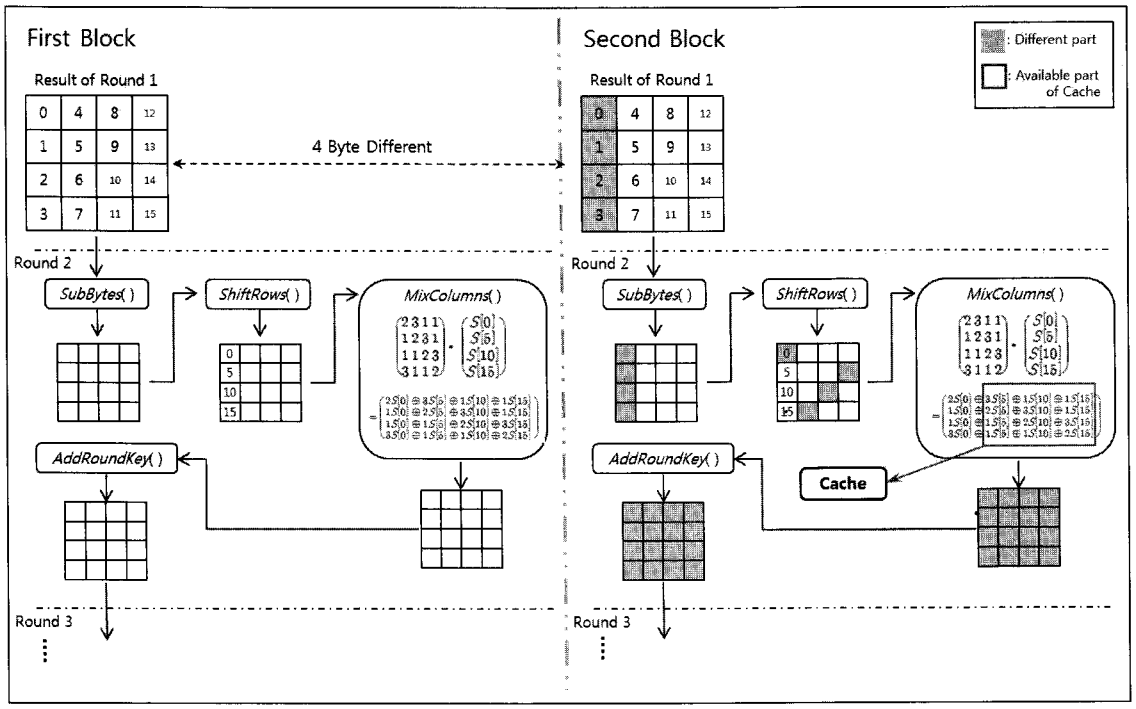
첫 번째 라운드를 거친 State는 두 번째 라운드의 입력으로 사용된다. 〔그림 5〕는 첫 번째 블록과 두 번째 블록의 Round 2 과정을 나타낸 것으로, 특히 *MixColumns*()에 *State*[0]의 변환 과정이 나타나 있다.

〔그림 5〕와 같이 첫 번째 블록과 두 번째 블록의 Round 1의 결과로 출력된 *State*는 *State*[0]만 다른 값을 갖는다. *State*의 각 바이트를 S[i] (i = 0, 1, 2, ... , 15) 라 표현하면, Round 2 과정에서 *ShiftRows()* 변환을 거친 후의 *State*[0]는 S[0], S[5], S[10], S[15] 바이트들로 구성된다. 이 *State*[0] 는 *MixColumns*()에서 아래와 같은 변환이 일어난다.

$$\begin{pmatrix}2&3&1&1\\1&2&3&1\\1&1&2&3\\3&1&1&2\end{pmatrix}\cdot\begin{pmatrix}S[0]\\S[5]\\S[10]\\S[15]\end{pmatrix}=\begin{pmatrix}rlt[0]\\rlt[1]\\rlt[2]\\rlt[3]\end{pmatrix}$$

$$=\begin{pmatrix}2S[0]\oplus3S[5]\oplus1S[10]\oplus1S[15]\\1S[0]\oplus2S[5]\oplus3S[10]\oplus1S[15]\\1S[0]\oplus1S[5]\oplus2S[10]\oplus3S[15]\\3S[0]\oplus1S[5]\oplus1S[10]\oplus2S[15]\end{pmatrix}$$

첫 번째 블록과 두 번째 블록에서 이 변환을 살펴보면, 두 블록의 각 *State*[0]에서 서로 다른 바이트는 S[0] 하나뿐이다. 따라서 위 연산에서 S[0]를 제외한 나머지 연산, 즉 rlt[0]의 (3·S[5] ⊕ 1·S[10] ⊕ 1·S[15]) 과 rlt[1]의 (2·S[5] ⊕ 3·S[10] ⊕ 1·S[15]), rlt[2]의 (1·S[5] ⊕ 2·S[10] ⊕ 3·S[15]), 그리고 rlt[3]의 (1·S[5] ⊕ 1·S[10] ⊕ 2·S[15]) 부분의 연산 결과를 캐시하여 사용할 수 있다. 이 부분의 연산 결과를 캐시하여 사용함으로써 3번의 XOR연산을 1번으로 줄일 수 있다. *State*[1], *State*[2], *State*[3] 에서도 역시 마찬가



〔그림 5〕 첫 블록과 두 번째 블록의 Round 2 과정

지로 각각 S〔3〕, S〔2〕, S〔1〕을 제외한 나머지 부분의 연산 결과를 캐시하여 사용할 수 있다.

본 장의 3.2 과 마찬가지로 Round 2의 입력 *State*의 *State*〔1〕, *State*〔2〕, *State*〔3〕의 값이 유지되는 동안에만 캐시된 정보가 사용될 수 있기 때문에 캐시된 정보가 사용될 수 있는 최대 블록의 수는 Round 1의 경우와 마찬가지로 2^8-1 이다.

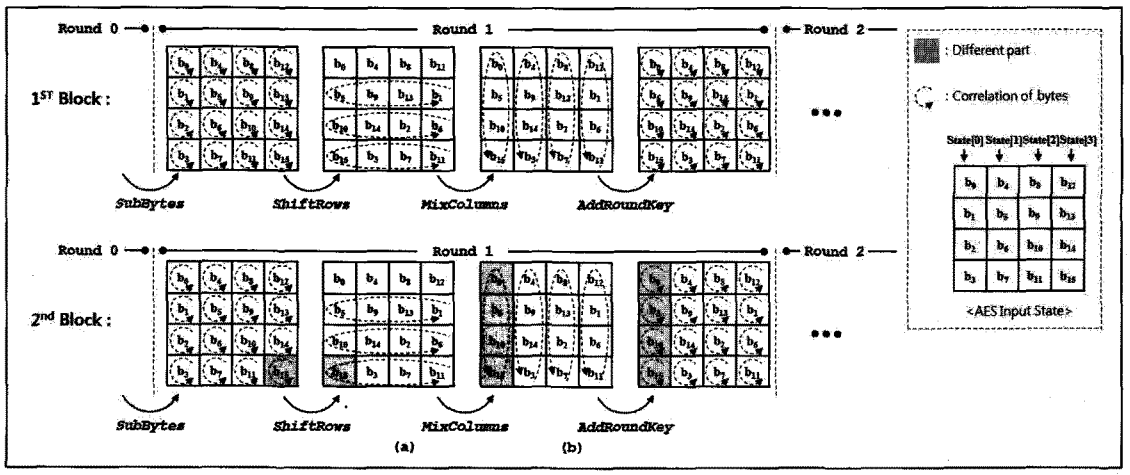
3.4 Pre-Computation of MixColumns for 1st Round (AES-CTR-C_rd$_{1+}$)

일반적으로 암호 모듈이 구현될 경우 두 단계로 나뉘어 구성된다. 첫 번째 단계는 *초기화 단계*이다. 암호 초기화 단계에서는 사용하려는 암호 알고리즘의 인스턴스가 사용하게 될 암호화 키와 IV가 결정된다. 이 단계에서 Key는 정해진 AES 키 길이에 따라 라운드 키로 확장되며, CTR Mode에서는 IV가 해당 암호 인스턴스의 Counter로 사용된다. 두 번째 단계는 *암/복호화 단계*로서 정해진 라운드 키와 IV를 사용하여 입력되는 입력 데이터를 암/복호화 하는 과정이 진행된다. 특히, 이 단계는 제공되는 암호 함수의 성격에 따라 입력을 불연속적으로 처리할 수 있도록 여러 과정을 거쳐 나뉘어서 진행되기도 하고, 따라서 같은 키와 IV라도 여러 번 반복적으로 호출될 수도 있다〔15〕. 본 장에서는 초기 1회만 호출되어 인스턴스를 유지하는 *초기화 단계*에서 AES CTR Mode의 Round 1에서 빈번하게 사용되는 정보를 미리 계산하여, 최대 4바이트 정보가 바뀌기 전까지 재사용 가능한 기법을 소개한다.

〔그림 6〕 는 AES CTR Mode가 동작하는 동안 Round 1에서의 데이터들 위치 변화와 상관관계가 미치는 영향을 나타낸다. AES 알고리즘의 입력은 다음과 같이 표현될 수 있다.

$$\begin{aligned} &Input \\ &= \{State[0], State[1], State[2], State[3]\} \\ &= \{b_0, b_1, \ldots, b_{15}\} \end{aligned}$$

암호의 입력 16 바이트의 데이터가 0부터 15까지 순서를 가진다고 했을 때, AES CTR Mode는 16번째 바이트 b_{15}부터 1씩 증가하여 입력을 매 블록마다 달리한다. 이때 두 번째 블록에서 b_{15}는 첫 번째 블록과 다르기 때문에 *MixColumns*() 연산에서 b_{15}는 b_0, b_5, b_{10}에 영향을 미치게 된다. b_0, b_5, b_{10}가 이전 블록과 비교했을 때 값의 변경이 없다는 점을 상기해보면, b_{15}의 변화가 Carry를 수반하지 않는다고 가정하면, b_{15}가 Round 1의 *MixColumns*() 연산에서 b_0, b_5, b_{10}



〔그림 6〕 Round 1의 State의 변환 과정

를 결정한다. 이는 〔그림 6〕에서 (a) *MixColumns*() 연산과 (b) *AddRoundKey*() 연산이 매 라운드마다 반복되는 것을 의미하고 따라서 미리 계산해놓은 (a)와 (b)를 재사용하는 방법을 적용하면 연산 효율성을 향상시킬 수 있다. 주어진 b_{15}에 대한 *State*〔0〕를 결정하는 테이블을 구성한다고 할 때, b_{15}는 0x00부터 0xFF 까지 변화하고 *State*〔0〕는 4 바이트이기 때문에 총 4 x 256 = 1 KBytes의 메모리가 필요하다. 이 테이블은 또한 모든 암호 인스턴스마다 같지는 않지만, 오직 입력된 키와 IV에 의존하기 때문에 *초기화 단계*에서 미리 계산하고 결정할 수 있다. *초기화 단계*는 매 암호 인스턴스에서 단 한번만 수행되기 때문에 실제 암/복호화 연산 단계에서 효율성을 확보할 수 있다.

생성된 치환테이블을 사용할 수 있는 것은 b_{15} 증가의 영향이 b_{10}에 영향을 미칠 때, 즉 $[b_{11}-b_{15}]$가 〔0xFFFFFFFFFF〕가 되어 b_{10}에 Carry가 발생할 경우이다. 이때 미리 계산된 치환 테이블에 업데이트 과정이 필요한데, 한번 업데이트가 이루어진 이후에는 $2^{40}\times16$ 바이트의 입력을 처리하는 동안에는 추가적인 테이블 업데이트 과정 없이 치환 테이블을 사용할 수 있다. 즉, 16 TeraBytes 의 데이터를 처리하는 동안에는 추가 연산과정이 필요 없기 때문에 연산 효율성을 높일 수 있다.

IV. 성능 분석

효율성 분석을 위해 *OpenSSL*〔16〕의 AES 소스 코드를 바탕으로 스마트 폰에서 성능을 측정해보았다. 테스트에 사용한 기기는 HTC社의 HD2 로서 1Ghz Qualcomm Snapdragon CPU, 448MB RAM, Windows Mobile 6.5 운영체제를 사용한다. AES-CTR-C_Ver1은 본 논문의 Ⅲ장의 3.1, 3.2를 구현한 것이고, AES-CTR-C_Ver2는 Ⅲ장의 3.1, 3.2, 3.3을 적용하여 구현한 것이며, AES-CTR-C_All는 본 논문에서 제안한 방법을 모두 적용하여 구현한 것이다.

테스트를 위해 AES-128, AES-192, AES-256 각각에 대해 초당 처리량(KBytes/s)과 한 블록의 처리시간(ms/1call)을 측정하였다.

초당 처리량은 한 번의 암호화 함수 호출시 입력되

〔표 1〕 1 Block 처리 시간(ms/1call)

	AES-128	AES-192	AES-256
AES-CTR-C_All	0.00069 ms	0.00084 ms	0.00098 ms
AES-CTR-C_Ver2	0.00071 ms	0.00086 ms	0.00101 ms
AES-CTR-C_Ver1	0.00075 ms	0.00089 ms	0.00104 ms
AES-CTR-Default	0.00093 ms	0.00108 ms	0.00121 ms
AES-CTR-C_All 시간단축율	26.15%	22.42%	18.90%
AES-CTR-C_Ver2 시간단축율	23.93%	20.85%	17.26%
AES-CTR-C_Ver1 시간단축율	19.94%	17.53%	14.54%

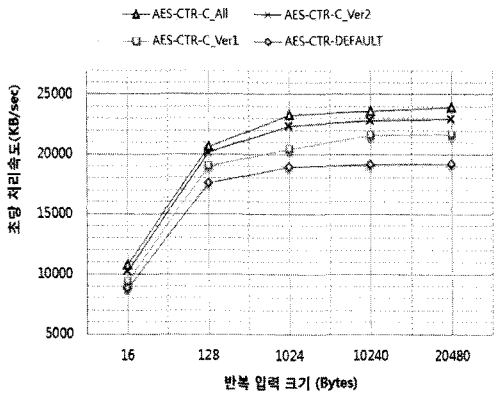
〔표 2〕 Key Schedule 처리 시간(ms)

	AES-128	AES-192	AES-256
AES-CTR-C_All	0.00049 ms	0.00054 ms	0.00058 ms
AES-CTR-C_Ver2	0.00048 ms	0.00053 ms	0.00057 ms
AES-CTR-C_Ver1	0.00048 ms	0.00053 ms	0.00057 ms

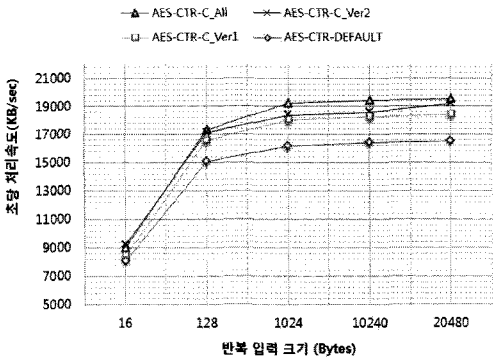
는 평문의 길이가 16, 128, 1024, 10240, 20480 Bytes일 때, 각각에 대해 초당 처리할 수 있는 바이트의 크기를 측정하였으며, 한 블록의 처리시간은 100만 블록에 대해 수행시간을 측정한 후 그 결과를 100만으로 나눈 값이다.

AES-128에서 입력 평문의 길이가 16 Bytes일 경우 AES-CTR-C_All의 처리량은 약 10.7 MB/s 로 기존의 AES 구현에 비해 22.30%, 128 Bytes는 약 20.6 MB/s 로 17.63%, 1024 Bytes는 약 23.3 MB/s 로 23.06%, 10240 Byte는 약 23.6 MB/s 로 23.15%, 20480 Bytes는 약 23.9MB/s 로 24.62% 향상되었다. 한 블록의 평균 처리시간은 AES-CTR-C_All은 0.00069ms로 기존 AES의 0.00093ms에 비해 약 26.15%의 시간단축율을 보였다.

AES-192는 입력 평문의 길이가 16 Bytes일 경우 AES-CTR-C_All의 처리량이 약 9.0 MB/s 로 기존의 AES 구현에 비해 11.56%, 128 Bytes는 약 17.3 MB/s 로 15.09%, 1024 Bytes는 약 19.2 MB/s 로 18.79%, 10240 Byte는 약 19.4 MB/s 로 18.17%, 20480 Bytes는 약 19.5 MB/s 로



〔그림 7〕 AES-128 초당 처리량 비교

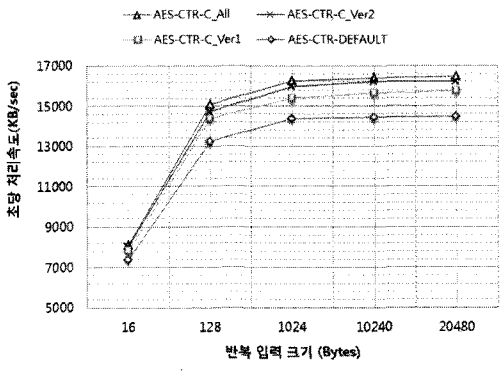


〔그림 8〕 AES-192 초당 처리량 비교

18.21% 향상되었다. 한 블록의 평균 처리시간은 0.00084ms로 기존 AES의 0.00108ms에 비해 약 22.42%의 시간단축율을 보였다.

AES-256은 입력 평문의 길이가 16 Bytes일 경우 AES-CTR-C_All의 처리량이 약 8.1 MB/s 로 기존의 AES 구현에 비해 9.48%, 128 Bytes는 약 15.0 MB/s 로 13.78%, 1024 Bytes는 약 16.2 MB/s 로 12.98%, 10240 Byte는 약 16.4 MB/s 로 13.54%, 20480 Bytes는 약 16.5 MB/s로 13.57% 향상되었다. 한 블록의 평균 처리시간은 0.00098ms로 기존 AES의 0.00121ms에 비해 약 18.90%의 시간단축율을 보였다.

본 논문 Ⅲ장의 3.4를 적용하는 경우 알고리즘의 초기화 단계에서 치환 테이블을 구성하게 된다. 〔표 2〕의 결과는 이러한 초기화 단계의 1000만번 반복 수행시간을 측정한 후 그 결과를 1000만으로 나눈 값이다. AES-128, AES-192, AES-256 모두 3.4 과정을 적용하기 전과 비교하여 약 0.00001 ms 정도의



〔그림 9〕 AES-256 초당 처리량 비교

시간이 추가로 소요된다. 따라서 초기 1회만 호출되는 초기화 과정에서 치환 테이블을 구성하는 작업은 암호 알고리즘의 전체 속도에 크게 영향을 주지 않음을 확인할 수 있다. 따라서 적은 오버헤드로 전체 알고리즘 연산의 효율성을 높일 수 있게 된다.

결과적으로 라운드 수가 가장 적은 AES-128이 본 논문에서 제안한 방법을 사용하였을 때 가장 높은 상승폭을 보였다.

V. 결 론

AES-CTR Mode는 병렬처리가 가능하다는 특성으로 인해 고속 처리를 필요로 하는 다양한 스트리밍 서비스 등의 암/복호화 연산에 사용되고 있다. IPTV, VoIP등의 서비스를 사용하는 모바일 환경은 자원이 제한되어 있기 때문에 이러한 환경에서 사용되는 암호화 알고리즘은 효율성을 고려하여 구현되어야 한다. 플랫폼에 특화된 알고리즘 성능 개선은 확장성이 많이 부족하지만, AES 알고리즘 자체의 구현 로직 개선은 알고리즘이 동작하는 아키텍쳐와 구현언어에 제한되지 않고 적용 가능하다.

본 논문에서는 경량화 된 환경에서 Cache를 이용하여 AES 알고리즘의 운영모드 중 CTR Mode의 효율성을 높일 수 있는 방법인 AES-CTR-C를 제안하였다. 블록의 처리가 진행되는 동안 매 암호 알고리즘의 입력이 작은 변화를 갖게 되는 카운터의 특성을 이용하면 AES 알고리즘의 처리속도를 1라운드 정도 감소시키는 효율성을 갖는 구현 기법을 구성할 수 있다. 향후, 1000Mbps의 기가비트 이더넷이나 1~7 Gbit/s 의 전송속도를 갖는 WiGig (Wireless Gigabit Alliance) 1.0[17] 네트워크 환경이 상용화되고 USB 3.0과 SDXC (Secure Digital eXtended Capacity) 메모리 등 고속의 처리 및 전송을 지원할 수 있는 환경이 도래하면 모바일 기기들은 보다 많은 암호화 처리량을 수행해야 하고 제안하는 기법과 같은 암호 모듈의 효율적인 구현 기법들이 필요할 것이다.

참고문헌

[1] NIST, "Advanced Encryption Standard (AES)", FIPS PUB 197, Nov. 2001.

[2] OMA, "DRM Content Format V2.0", Apr. 2004.

[3] 이진홍, 이혜주, 신상욱, "방송 콘텐츠를 위한 안전한 유통 시스템 설계 및 구현", 한국정보보호학회논문지, 17(2), pp. 19-27, 2007년 4월.

[4] IETF, "The Secure Real-time Transport Protocol(SRTP)", RFC3711, Mar. 2004.

[5] Helger Lipmaa, Phillip Rogaway and David Wagner, "Comments to NIST Concerning AES-modes of Operation : CTR-mode Encryption", [Online]. Available : http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ctr/ctr-spec.pdf, Oct. 2000.

[6] NIST, "Recommendation for Block Cipher Modes of Operation : Methods and Techniques", SP 800-38A, Dec. 2001.

[7] Gael Rouvroy, Francois-Xavier Standaert, Jean-Jacques Quisquater and Jean-Didier Legat, "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications", Information Technology: Coding and Computing, Vol.2, pp. 583-587, Apr. 2004.

[8] Nazar A.Saqib, Francisco Rodriguez-Henriquez and Arturo Diaz-Perez, "AES Algorithm Implementation - An efficient approach for Sequential and Pipeline Architectures", Computer Science 2003, pp. 126-130, Sep. 2003.

[9] Francois Charot, Eslam Yahya and Charles Wagner, "Efficient Modular-Pipelined AES Implementation in Counter Mode on ALTERA FPGA", Computer Science 2003, Vol.2778, pp. 282-291, 2003

[10] 정창호, 박일환, "64-비트 프로세서에서 AES 고속 구현", 한국정보보호학회논문지, 18(6A), pp. 51-61, 2008년 12월.

[11] Guido Bertoni, Luca Breveglieri, Pasqualina Fragneto, Marco Macchetti, and Stefano Marchesin, "Efficient Software Implementation of AES on 32-Bit Platform", Proceedings of CHES'02, volume 2523 of Lecture Notes in Computer

Science, pp. 129-142, 2003

[12] Daniel J. Bernstein and Peter Schwabe, "New AES software speed records", INDOCRYPT '08 Proceedings of the 9th International Conference on Cryptology in India: Progress in Cryptology, pp. 322-336, Sep. 2008.

[13] CRYPTO++ Library, [Online]. Available : http://www.cryptopp.com

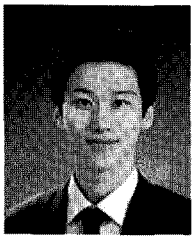
[14] Hongjun Wu, eSTREAM Project, [Online]. Available : http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/ecrypt/trunk/benchmarks/aes-ctr/aes-128/hongjun/v1/?rev=203#dirlist

[15] RSA Laboratories, "PKCS#11:Cryptographic Token Interface Standard", [Online]. Available : ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf

[16] The OpenSSL Project, [Online]. Available : http://www.openssl.org

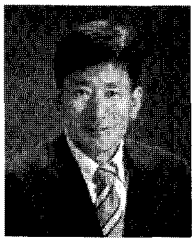
[17] Wireless Gigabit Alliance, "WiGig White Paper : Defining the Future of Multi-Gigabit Wireless Communications" , [Online]. Available : http://www.wigig.org, Jul. 2010.

〈著者紹介〉



박 진 형 (Jin Hyung Park) 학생회원
2010년 2월: 건국대학교 컴퓨터공학과 졸업
2010년 2월 ~ 현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 암호프로토콜, 스마트폰 보안, 암호알고리즘

백 정 하 (Jung Ha Paik) 학생회원
2006년 2월: 고려대학교 수학과 졸업
2006년 2월 ~ 2008년 2월: 고려대학교 정보경영공학전문대학원 석사
2008년 3월 ~ 현재: 고려대학교 정보보호대학원 박사과정
〈관심분야〉 암호프로토콜, VANET, 스마트폰 보안, 클라우드 컴퓨팅, 애드 혹 네트워킹



이 동 훈 (Dong Hoon Lee) 종신회원
1983년 8월: 고려대학교 경제학사 졸업
1987년 12월: Oklahoma University 전산학과 석사 졸업
1992년 5월: Oklahoma University 전산학과 박사 졸업
1993년 3월 ~ 1997년 2월: 고려대학교 전산학과 조교수
1997년 3월 ~ 2001년 2월: 고려대학교 전산학과 부교수
2001년 3월 ~ 현재: 고려대학교 정보보호대학원 교수
〈관심분야〉 암호프로토콜, 암호이론, USN이론, 키 교환, 익명성 연구, PET 기술