CPU기반 DES, RC4, Blowfish 그리고 MD5 암호 알고리즘 최적화 벤치마크

엄시우*, 김현준*, 권혁동*, 서화정**

*한성대학교 (대학원생)
**한성대학교 (교수)

CPU-based DES, RC4, Blowfish, and MD5 Encryption Algorithm Optimization Benchmarks

Si-Woo Eum*, Hyun-Jun Kim*, Hyeok-Dong Kwon*, Hwa-Jeong Seo**

*Hansung University(Graduate student)
*Hansung University(Professor)

요 약

DES, RC4, MD5 그리고 Blowfish 암호 알고리즘은 오래된 암호이지만 여전히 여러 곳에서 사용되고 있는 암호 알고리즘이다. 오랜 최적화 연구로 최적화된 구현이 널리 사용되고 있다. 이로 인해 여러 오픈 소스로 공개된 구현 간의 큰 차이점을 발견하기 어렵다. 하지만 약간의 차이로 성능 차이가 발생한다. 본 논문에서는 오픈 소스로 공개된 구현 코드를 조사하고 성능을 측정하여 각 암호 알고리즘에 대하여 최적화된 성능을 제시하며 성능 차이가 발생한 원인에 대해 조사한다.

I. 서론

DES, RC4, MD5, Blowfish 암호 알고리즘은 오래된 암호이지만 여전히 여러곳에서 사용되고 있는 암호 알고리즘이다. 오래된 암호이기에 오랜 최적화 연구로 최적화된 구현이 널리 사용되고 있다. 오픈 소스로 공개된 각각의 구현들의 경우에도 구현 코드를 보면 대부분 비슷한 구현을 보여주고 있으나, 약간의 구현 차이가 존재한다.

본 논문에서는 각 암호들에 대하여 오픈 소스로 공개된 코드들을 조사하고 성능 비교를 통해 최적화된 성능을 보여준다.

II. 관련 연구

2.1 DES

DES 블록 암호는 1979년에 미국 NIST(National Institute of Standards and Technology)에서 국가 표준 암호 알고리즘으로 지정한 대칭키 암호 알고리즘이다.

DES의 블록과 키 길이는 64-bit를 지원하며, 64-bit 키 중에서 8-bit는 parity bit로 사용된다. Feistel 구조로 16번 반복하며 암호화가 진행된다[1].

2.2 RC4

RC4는 초경량 스트림 암호 알고리즘으로 내부 구조가 간단하여 빠르게 암호화가 가능한 암호이다. IEEE 802.11의 무선랜 보안 프로토콜 WEP, 오라클의 SQL 보안, IEEE 802.11i의 TKIP등에 사용되고 있는 암호이다.

RC4 알고리즘은 가변적 키 길이를 지원하며, 키를 통해 상태 배열(Sbox)를 섞어주는 KSA 과정과 암호문을 생성하는 PRGA 과정으로 나뉜다[2].

2.3 Blowfish

Blowfish 블록 암호는 1993년 Bruce Schneier가 설계한 대칭키 암호 알고리즘이다.

Blowfish의 블록 길이는 64-bit이며, 키 길이는 32-bit에서 448-bit 범위의 가변 키 길이를 지원한다. Feistel 구조로 16번 반복하며 암호화가 진행된다[3].

2.4 MD5

MD5는 128-bit 해시 함수 알고리즘이다. MD5는 임의의 길이를 입력받아 128-bit 크기의 해시 값을 출력한다. 입력된 값은 512-bit 블록으로 나뉘어 연산되며, 512-bit로 나뉠 수 있도록 패딩된다. 64번 반복되며 암호화를 진행하며, 이때 각 라운드에서 사용되는 F 함수는 라운드마다 다른 연산으로 구현되어 있다[4].

2.5 OpenSSL

OpenSSL은 Eric A. Young, Tim J. Hudson에 의해 TLS, SSL, DTLS 프로토콜을 C 언어로 구현한 오픈 소스 통신 라이브러리이다. 윈도우와 리눅스, MacOS 모두 설치가 가능하다.

OpenSSL에서는 다양한 암호 알고리즘이 구현되어 있을 뿐만 아니라 Speed 명령어를 통해서 구현된 암호 알고리즘의 성능을 확인할 수 있다. 본 논문에서는 OpenSSL에서 제공하는 Speed 명령어로 측정된 성능과 OpenSSL에서 구현된 암호 알고리즘을 따로 분리해 Standalone으로 동작한 성능을 비교 지표로 확인한다[5, 6].

III. 코드 분석

3.1 DES

DES의 최적화된 구현은 Bitslicing 기법을 사용한 DES 구현이다. Bitslicing 기법은 [7]에서 처음 제안한 기법이다. 하드웨어 논리 회로를 구현하는 것처럼 단일 비트 논리 연산을 함수로 표현한다. CPU에서 비트 연산을 사용하여 병렬로 함수의 여러 인스턴스를 수행하며, 속도 병렬화, 일정한 실행 시간이 장점인 구현 기법이다. Bitslicing 기법은 구현되는 환경의 레지스터 크기 만큼 여러 블록을 저장하여 병렬로 수행 가능하다. 본 논문에서는 오픈 소스로 공개된 Bitslicing 기법을 활용한 Matthew Kwan의 DES 구현 코드를 활용하여 성능 측정을 하였다[8].

3.2 RC4

RC4 최적화 구현 오픈 소스로는 Apple의 오픈 소스 코드[9]와 Nayuki[10]가 있다. Nayuki 오픈 소스의 경우 C 구현뿐만 아니라 x86-64 어셈블리로 구현된 코드도 제공하고 있다. RC4 알고리즘은 구조가 단순하여 OpenSSL을 포함한 3개의 오픈 소스 RC4 구현 코드의 차이는 크지 않다. 하지만 차이점으로 볼 수 있는 점은 Apple 코드의 경우 Swap 연산을 함수로 구현하여 함수 호출을 통해 Swap 연산을 하며, Nayuki 코드의 경우 Swap 연산을 함수나 매크로로 구현하지 않고 풀어서 구현하였다. OpenSSL의 경우 전체 연산을 매크로로 구현하였다. 또한 Apple과 Nayuki 구현은 1-byte 단위로 반복되며 암호화가 진행되는데, OpenSSL 구현은 8-byte 단위로 반복하며 암호화가 진행된다. 따라서 입력문의 크기가 8-byte로 안떨어지면, 조건문을 사용하여 나머지 값(1~7-byte)에 대해서는 1-byte 단위로 연산된다.

3.3 Blowfish

Blowfish는 OpenMP를 활용한 오픈 소스[11]와 OpenSSL의 비교를 진행한다. Blowfish의 알고리즘은 RC4와 마찬가지로 단순한 연산으로 설계되어 있다. 따라서 OpenMP를 활용한 오픈 소스와 OpenSSL의 구현 차이는 크게 없다. 하지만 OpenSSL에서는 OpenMP를 사용하지 않기 때문에 OpenMP를 활용한 구현과의 성능 비교를 진행한다.

OpenMP(Open Multi-Processing)는 공유 메모리 다중 처리 프로그래밍 API로 CPU에서 제공하는 멀티 코어를 통해 병렬 구현이 가능하다. 개발자가 병렬로 실행될 코드의 블록만을 간단하게 설정하고, 세밀한 제어는 컴파일이나 런타임 시스템에게 맡김으로써 간단하게 CPU의 멀티 코어를 활용한 병렬 구현을 할 수 있다.

3.4 MD5

RC4와 마찬가지로 Nayuki의 오픈 소스 코드[12]와 비교를 진행한다. 오래된 암호로 최적화된 코드가 많이 알려져 있어 두 구현의 차이는 크지 않다. MD5는 라운드에 따른 F 함수의 연산이 다르다. 공통적으로 두 구현 모두 매크로로

각 라운드에 따른 F 함수를 구현하여 사용한다.

하지만 OpenSSL의 경우 C로 구현된 MD5가 있으나, 몇 개의 환경에 대해서는 Assembly로 구현되어 있다. 따라서 본 논문의 구현 환경인 x86-64에서는 OpenSSL의 MD5는 Assembly로 구현된 코드가 동작된다.

IV. 성능 평가

성능 측정은 MacBook Pro(15-inch, 2018), 2.6Ghz 6코어 Intel Core i7상에서 진행한다. GCC 버전은 Apple Clang version 14.0.0이며, OpenSSL 버전은 1.1.1h 버전을 사용하였다.

OpenSSL의 Speed 명령어는 3초 동안 암호화 횟수를 측정하여 성능을 제시하고 있다. 또한 3초 동안 암호화한 횟수를 통해 초당 암호화 크기(KB/s)를 제공하고 있으며 조사한 코드들도 같은 방법으로 성능을 측정하여 비교 분석한다.

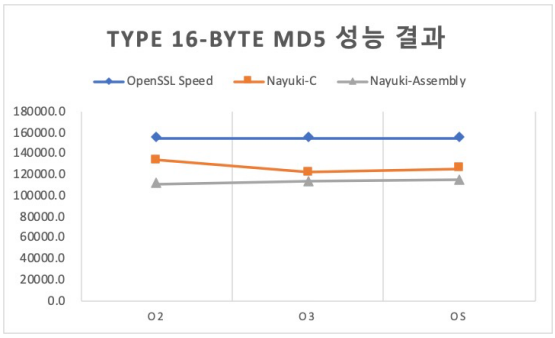

Fig. 1. MD5 Performance result of input length 16-byte

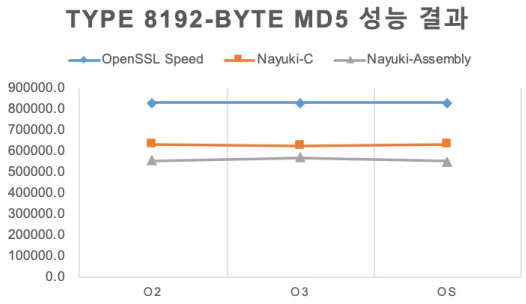

Fig. 2. MD5 Performance result of input length 8192-byte

MD5에서는 입력 길이에 상관 없이 OpenSSL에서의 성능이 가장 빠른 것으로 측정되었다. 따라서 Nayuki의 Assembly 코드 보다 OpenSSL에서 구현된 Assembly 코드가 더 최적화된 구현이다. 또한 OpenSSL의 구현은 x86-64 환경이 아닌 곳에서 동작시 C로 동작하게 되는데, 이 경우에는 Nayuki의 구현과 동일한 성능을 보여주었다.

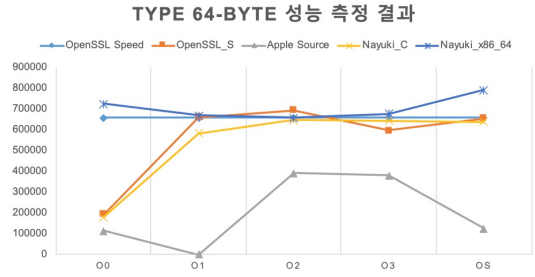

Fig. 3. RC4 Performance result of input length 64-byte

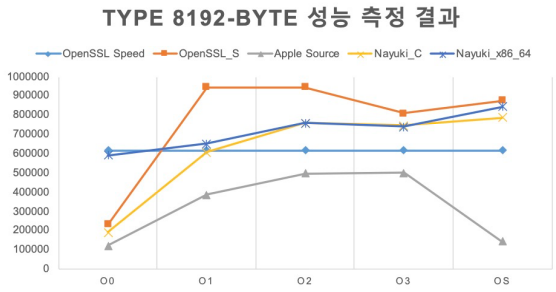

Fig. 4. RC4 Performance result of input length 64-byte

RC4에서는 약간의 구현 차이가 큰 성능 차이를 보여주는 것을 알 수 있다. 함수로 구현한 Apple 오픈 소스의 경우 데이터 길이와 상관없이 낮은 성능을 보여주고 있다. Nayuki 오픈 소스에서는 낮은 데이터에서 Assembly 구현 코드가 높은 성능을 보여주지만 OpenSSL과 큰 차이가 있지는 않다. 하지만 데이터가 커질수록 OpenSSL 구현 코드에서 높은 성능을 보여주고 있다. 이는 반복수의 차이에서 발생하는 성능 차이로 분석된다.

OpenSSL	Bitslice	Bitslice-omp
101,863	202,822	786,764

Table 2. DES Performance result of input length 256-byte(unit : KB/s, omp : using OpenMP)

Bitslice 구현에서는 64-bit 크기의 데이터 32개를 병렬 암호화하기 때문에 256-byte의 입력 크기에 따른 성능 비교를 진행하였다. 또한 블록 간에 영향을 주지 않기 때문에 OpenMP를 활용한 병렬 구현도 가능하다. 결과적으로 Bitslice 구현이 OpenSSL 구현보다 2배 높은 성능을 보여주고 있으며, OpenMP를 사용함으로써 3.8배 더 높은 성능을 보여준다.

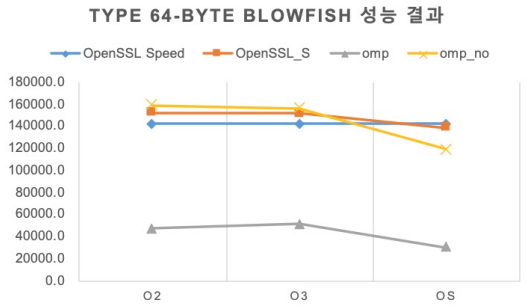

Fig. 5. Blowfish Performance result of input length 256-byte

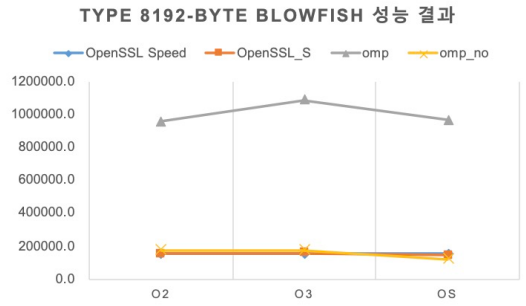

Fig. 6. Blowfish Performance result of input length 8192-byte

Blowfish에서는 함수 구현의 차이는 없지만 OpenMP를 사용함으로써 성능 향상된 결과를 확인할 수 있다. 낮은 데이터에서는 병렬 구현의 장점이 오히려 성능에 영향을 주고 있어 낮은 성능을 보여주지만, 데이터가 커질수록 병렬 구현으로 인해 성능이 향상되어 차이가 커진 것을 볼 수 있다. 결과적으로 OpenMP를 사용하는 것만으로도 큰 성능 향상을 얻을 수 있음을 확인할 수 있다.

V. 결 론

본 논문에서는 DES, RC4, Blowfish 그리고 MD5 암호 알고리즘의 오픈 소스 코드에 대한 최적화 구현을 조사하고 성능 비교를 진행하였다. 오래된 암호로 인해서 충분한 연구와 노하우로 구현 코드의 큰 차이는 없으나 약간의 구현 차이가 큰 성능 차이를 보여주고 있으며, 시간이 지나면서 발전한 기술을 적용함으로써 더욱 최적화된 구현이 가능함을 보여준다.

VI. Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00540, Development of Fast Design and Implementation of Cryptographic Algorithms based on GPU/ASIC, 100%).

[참고문헌]

[1] Matsui, Mitsuru. "Linear cryptanalysis method for DES cipher." Workshop on the Theory and Application of of Cryptographic Techniques. Springer, Berlin, Heidelberg, 1993.
[2] 이규희. "클러스터를 이용한 고성능 RC4 암호화 하드웨어 설계." 한국정보통신학회논문지 23.7 (2019): 875-880.
[3] Khatri-Valmik, Ms Neha, and V. K. Kshirsagar. "Blowfish algorithm." IOSR Journal of Computer Engineering (IOSR-JCE) 16.2 (2014): 80-83.
[4] Rivest, Ronald. The MD5 message-digest algorithm. No. rfc1321. 1992.
[5] OpenSSL. https://www.openssl.org/.
[6] 박기태, 한효준, and 이재훈. "OpenSSL 상에서 LEA 설계 및 구현." 한국통신학회논문지 39.12 (2014): 822-830.
[7] Biham, Eli. "A fast new DES implementation in software." International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, 1997.
[8] Matthew Kwan, “Bitslice DES,” http://www.darkside.com.au/bitslice/. Accessed: 2016-05-2.
[9] Apple. “rc.c,” https://opensource.apple.com/source/OpenSSL/OpenSSL-12/openssl/crypto/rc4

/rc4.c.auto.html
[10] Nayuki. “RC4 cipher in x86 assembly”, https://www.nayuki.io/page/rc4-cipher-in-x86-assembly
[11] Tuan Dao. “blowfish/omp”, https://github.com/tuan2195/blowfish
[12] Nayuki. “Fast MD5 hash implementation in x86 assembly”, https://www.nayuki.io/page/fast-md5-hash-implementation-in-x86-assembly