Smart Media Journal / Vol.14, No.8 / ISSN:2287-1322
https://dx.doi.org/10.30693/SMJ.2025.14.8.50

GPU 상에서의 PIPO 블록암호 CTR모드 최적 구현

(Optimal implementation of PIPO block cipher CTR mode on GPU)

송민호*, 엄시우*, 김상원**, 서화정***
(Min Ho Song, Si Woo Eum, Sang Won Kim, Hwa Jeong Seo)

요 약

본 논문에서는 GPU 환경에서 PIPO 블록 암호의 CTR 모드를 최적화하는 기법을 제안한다. 기존의 개별 블록 암호화 방식과 다르게 4개의 평문을 Packing하여 병렬로 처리하는 최적화 기법을 적용하여 연산 속도를 향상시켰다. 또한 CUDA의 내장 함수를 활용하여 평문과 마스터키의 Packing 및 Unpacking을 최적화하였으며 Sbox, Pbox, 그리고 KeyAddition 연산을 4개의 평문에 대해 동시에 수행함으로써 메모리 접근 횟수를 최소화하고 암호화 성능을 개선하였다. 이러한 최적화 기법을 적용한 암호화 방식은 기존 방식 대비 글로벌 메모리 접근 횟수를 줄이고 연산 성능을 개선하는 효과를 나타내었다.

■ 중심어 : GPU ; PIPO ; CTR 모드 ; 병렬 연산 ; 최적화 구현

Abstract

In this paper, we propose a technique to optimize the CTR mode of the PIPO block cipher in a GPU environment. Unlike the existing individual block encryption method, we improve the computation speed by applying an optimization technique that packs four plaintexts and processes them in parallel. In addition, we optimize the packing and unpacking of plaintexts and master keys by utilizing the built-in functions of CUDA, and minimize the number of memory accesses and improve the encryption performance by performing Sbox, Pbox, and KeyAddition operations on four plaintexts simultaneously. The encryption method applying this optimization technique showed the effect of reducing the number of global memory accesses and improving the computational performance compared to the existing method.

■ keywords : GPU ; PIPO ; Counter Mode ; Parallel Processing ; Optimal Implementation

Ⅰ. 서 론

사물 인터넷(IoT) 등 자원이 제한적인 환경에서는 경량 암호 기법의 필요성이 크게 대두되고 있다[1]. AES와 같은 기존의 대칭키 블록 암호 알고리즘은 이러한 환경에서 연산 성능이나 자원 소모 측면에서 비효율적일 수 있으며, 이를 보완하기 위해 지난 수십 년간 다양한 경량 블록 암호들이 개발·제안되어 왔다. 전통적인 CPU 기반 암호 솔루션은 대규모 실시간 서비스 요구 사항을 충족하는 데 한계를 보인다. 이에 대응하여

* 준회원, 한성대학교 정보컴퓨터공학과
** 준회원, 한성대학교 융합보안학과
** 정회원, 한성대학교 융합보안학과
This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services, 50%) and this work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2025-02306395, Development and Demonstration of PQC-Based Joint Certificate PKI Technology, 50%)
접수일자 : 2025년 03월 27일 게재확정일 : 2025년 07월 03일
수정일자 : 2025년 05월 19일
교신저자 : 서화정 e-mail : hwajeong84@gmail.com

그래픽 처리 장치(GPU)는 강력한 암호 가속기로 부상했으며, 수백에서 수천 개의 코어가 동시에 작동하는 대규모 병렬 처리를 제공한다[2]. 지난 10년 동안의 학술 연구는 GPU를 활용한 암호화의 가능성을 지속적으로 입증하며, 놀라운 처리량 향상을 달성해 왔다.

예를 들어, 2007년 NVIDIA Geforce 7900 GT를 사용한 AES 암호화 구현은 약 0.85GB/s(약 6.8Gb/s)의 성능을 기록하며 당시 CPU보다 뛰어난 속도를 보였다[3]. 이후 최적화된 GPU 기반 블록 암호 구현은 수십 기가비트에서 수백 테라비트 수준까지 도달했다. 이러한 발전은 GPU 하드웨어의 지속적인 개선과 더불어 연구자들이 개발한 고급 병렬화 기법 덕분이다.

특히, 경량 블록 암호인 CHAM(64비트 블록 암호)은 NVIDIA RTX 2070 GPU에서 3.03Tbps의 처리량을 기록하며 경량 암호화 알고리즘이 GPU 아키텍처를 활용할 경우 놀라운 성능을 발휘할 수 있음을 입증했다. AES의 경우도 유사한 조건에서 300Gbps 이상의 성능을 달성하며 최적화된 GPU 기반 구현의 잠재력을 보여준다.

이러한 맥락에서 PIPO 블록 암호는 제약된 환경에서 보안성과 고속 암호화를 제공할 수 있는 유망한 후보로 주목받고 있다. 따라서 본 연구는 PIPO 블록 암호 CTR 모드를 GPU 환경에서 고속으로 병렬 처리하기 위한 최적화 기법을 제안하고 이를 구현하여 성능을 평가하는 것을 목적으로 한다. 다양한 GPU 아키텍처에서의 성능 비교를 통해 제안 기법의 효율성과 적용 가능성을 검증하고자 한다.

Ⅱ. 관련 연구

1. PIPO Algorithm

2020년 ICISC 학회에서 부채널 공격(Side-Channel Attack) 대응용 보호 기법 적용 시 발생하는 성능 오버헤드를 최소화하기 위한 새로운 경량 블록 암호 PIPO가 제안되었다[4]. PIPO(“Plug-In”과 “Plug-Out”)라는 이름은 각각 부채널 보호가 적용된 환경과 적용되지 않은 환경에서의 활용을 의미한다. 이 알고리즘은 64비트 블록 크기와 128비트 또는 256비트 키를 지원하는 SPN(Substitution-Permutation Network) 구조로 설계되었으며, 8비트 AVR 마이크로컨트롤러와 같은 플랫폼에서 효율적인 비트슬라이스 구현이 가능하도록 바이트 지향적인 구조를 채택하고 있다. 또한 비선형 연산 횟수를 최소화함으로써 고차 마스킹 등 부채널 공격 대비 기법을 적용하더라도 성능 오버헤드가 매우 적다는 특징을 갖는다[5]. 그 결과 PIPO는 동일한 블록 및 키 크기를 갖는 기존 블록 암호들보다도 측채널 보호 적용 여부와 무관하게 전반적으로 우수한 소프트웨어 구현 성능을 보이는 것으로 보고되었다[6].

PIPO 알고리즘의 내부 연산은 S-Layer, R-Layer, AddRoundKey의 세 단계로 구성된다. S-Layer는 8비트 Sbox를 기반으로 하며 8비트 입력에 대해 8비트 출력을 생성하는 변환 과정이다. 이 연산은 TLU 방식을 이용하거나 비트슬라이싱 방식으로 구현될 수 있다. R-Layer는 8비트 단위로 로테이션 과정을 수행한다. 이러한 S-Layer와 R-Layer 연산에 라운드 키가 추가되는 AddRoundKey 연산을 결합하여 하나의 라운드를 구성하며 전체 암호화는 PIPO-64/128의 경우 13라운드, PIPO-64/256의 경우 17라운드에 걸쳐 수행된다.

2. CTR 모드

CTR(Counter Mode, 카운터 모드)는 블록 암호를 스트림 암호처럼 동작하도록 하는 블록 암호 운용 모드 중 하나이다. 평문을 여러 블록으로 나누고 각 블록 마다 서로 Counter 값과 동일한 Nonce 값을 합한 IV(Initial Vector, Nonce + Counter) 값을 암호화하여 키스트림을 생성하고, 이를 해당 Counter 값에 해당하는 평문 블록과 XOR하여 암호문을 생성한다. CTR 모드는 1979

년 Diffie와 Hellman에 의해 제안되었으며, 2001년 NIST에 의해 AES 표준의 운영 모드로 채택되었다[7].

CTR 모드의 암호화와 복호화 과정은 대칭적이다. 즉, 복호화 시에도 동일하게 해당 블록의 Counter 값을 사용한 IV 값을 암호화하여 키스트림을 생성하고 암호문과 XOR하면 원문으로 복호화된다. 즉, 암호화만으로 암·복호화가 모두 가능하다.

CTR 모드는 각 블록이 독립적으로 처리되므로 멀티코어 CPU나 병렬 연산에 특화된 장치 GPU에서 병렬 연산을 수행하기에 최적이다. 특히 수많은 코어를 가진 GPU에서 각각의 코어에서 각각의 블록을 독립적으로 처리함으로써 동시에 암호화가 가능하다. GPU를 활용하여 대용량을 암호화함으로써 CPU의 부하를 줄이고 전체 처리량을 크게 향상시킬 수 있다.

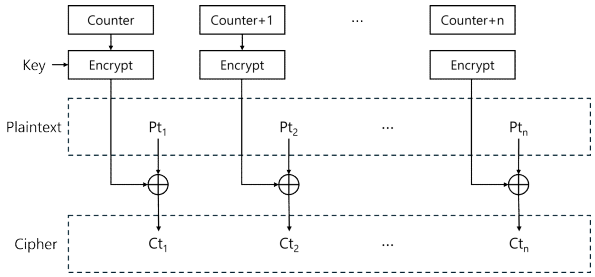

그림 1. CTR Mode Encryption Process

3. GPU 아키텍처

GPU(Graphic Processing Unit)는 그래픽 처리를 가속하기 위한 특수 목적을 가진 프로세서이다. 2000년대 중반 Nvidia의 CUDA와 같은 GPGPU(General Purpose GPU)가 등장하면서 그래픽 이외의 연산에도 GPU가 활용되기 시작했다. GPU가 범용 연산 능력을 갖추면서 인공지능 연산이나 암호화 연산과 같은 고도로 병렬적인 작업에 필수적인 요소로 자리 잡았다. GPU는 동시에 매우 많은 계산을 병렬 수행할 수 있는 구조 덕분에 병렬 처리에 적합한 문제에서 뛰어난 성능을 보여주며, CPU로만 수행되던 작업을 GPU로 가속화하는 연구가 활발히 진행되고 있다[8].

GPU는 다수의 연산 코어(Cluster)로 구성되며, Nvidia의 스트리밍 멀티프로세서(SM), AMD의 컴퓨트 유닛(CU) 등이 이에 해당한다. 하나의 SM/CU 안에는 수백 개의 연산용 ALU와 특수 연산 유닛이 배열되어 있고, 자체 레지스터 파일과 L1 캐시, 공유 메모리 등을 갖추고 있다. 즉, 각 코어에는 연산에 필요한 데이터와 명령을 빠르게 처리하기 위한 전용 자원과 다수의 스레드를 병렬 처리하기 위한 SIMD 연산 유닛 조합이 존재한다. 이러한 코어들이 한 GPU 내에 수십 개 이상 포함되어 병렬 작업을 수행함으로써 우수한 병렬 처리 능력을 보여준다[9].

GPU는 메모리 계층 구조를 통해 대용량의 데이터를 공급하는 데에 효율적으로 동작되도록 설계되어 있다. 가장 빠른 메모리는 각 스레드가 사용하는 레지스터이며, 다음으로는 각 SM 내부에 장착된 L1 캐시와 공유 메모리가 있다. 메모리의 접근을 빠르게 할 수 있지만 제한적으로 사용할 수 있다. 특히 공유 메모리는 프로그래머가 직접 관리할 수 있는 메모리로써 동일 블록의 스레드들이 데이터를 공동으로 사용할 수 있어 유용하게 사용할 수 있다. 이 외에도 여러 SM 코어가 함께 사용하는 상위 메모리 계층에는 L2 캐시와 글로벌 메모리가 있다. 메모리의 크기가 크지만 하위 계층의 메모리 보다 느린 속도를 보여준다[10, 11, 12]. 전체적인 GPU 아키텍처 구조는 그림 2.와 같다.

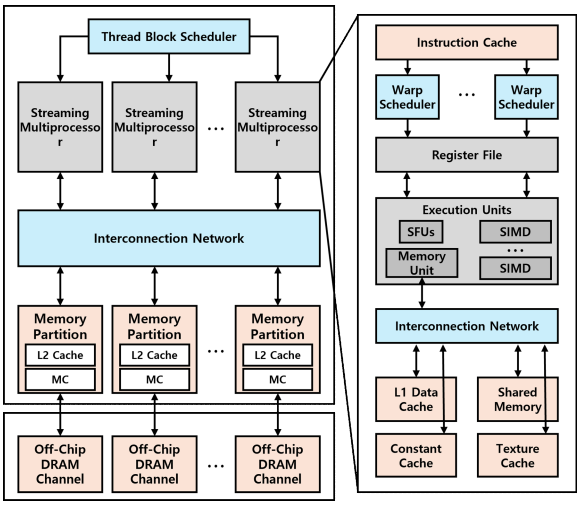

그림 2. Structure of GPU Architecture

GPU를 활용한 대칭키 암호의 가속화 연구는 2007년 NVIDIA CUDA와 같은 GPGPU의 개발과 함께, AES-CTR과 같은 블록 암호를 GPU에서 구현하는 다양한 연구가 수행되었다. 특히 Tezcan의 2021년 연구에 따르면, Nvidia RTX 2070 Super GPU 하나로 AES-128-CTR을 암호화할 때, 878.6Gps에 이르는 처리량을 달성하였다. 이는 기존의 GPU 최고 성능 대비 2.56배 향상된 성능을 보여주며, AES-NI 명령어를 활용한 최신 CPU보다 뛰어난 성능을 보여준다. 모바일용 저가형 GPU인 GeForce MX250에서도 AES-256-CTR 암호화에서 60Gbps 정도의 처리량을 보여주고 있으며 이는 일반적인 SSD 디스크의 읽기/쓰기 속도 수준을 넘는 성능이다. 다수의 GPU를 활용하면 테라비트(Tbps) 급의 처리량의 성능을 보여주는 연구도 수행되고 있으며[13] 이러한 연구 결과들은 실제로 CPU보다 GPU를 활용함에 있어 더 높은 성능을 보여줄 수 있다는 것이 확인되고 있다.

4. 이전 연구

기존 연구인 Choi et al.[14]에서는 AVX-2, AVX-512, GPU 환경에서의 PIPO 병렬 구현 기법을 제안하였다. PIPO 알고리즘의 S-Layer 및 R-Layer 연산은 모두 8비트 단위로 수행되며 이는 8비트 연산자를 지원하는 환경에서는 문제가 없지만 그렇지 않은 환경에서는 16/32/64비트 연산자를 사용해야 하므로 비효율적이다. AVX 명령어의 비트 시프트 최소 단위는 16비트이고 GPU PTX 명령어 역시 최소 16비트 단위로 연산을 수행한다. 이에 따라 해당 연구에서는 두 개의 8비트 평문을 16비트 데이터 형식으로 결합하여 처리하는 방식을 채택하였다.

S-Layer 연산은 비트 슬라이싱 기법을 적용하여 1비트 단위로 독립적으로 수행함으로써 두 개의 평문이 16비트 데이터 형식으로 저장되어 있어도 상호 간섭 없이 처리할 수 있다. 반면 R-Layer의 주요 연산인 비트 로테이션은 8비트 단위가 16비트 자료형에 저장된 상태에서 직접 적용 시 데이터가 섞여 정확한 값을 얻을 수 없는 문제가 발생한다. 이를 해결하기 위해 비트 마스킹 기법을 통해 로테이션 연산을 처리하였다.

한편 Eum et al.[15]에서는 32비트 RISC-V 환경에서 PIPO 최적 병렬 구현을 제안하였다. 이 연구에서는 4개의 입력 블록을 재배열하여 병렬 처리를 수행하였고 PIPO CTR 모드에서 사용하는 Nonce와 Counter를 8비트 단위로 분할한 뒤 32비트로 확장하여 레지스터 내부에서 정렬하는 방식을 적용하였다. Nonce는 기존 48비트에서 192비트로 확장하여 활용하고 Counter는 기존 16비트를 그대로 사용하며 현재 상태를 기준으로 연산을 수행하였다.

병렬 환경에서 S-Layer 연산 시 데이터가 섞일 수 있는 문에 대해서는 마스킹 기법을 통해 해결하였다. 라운드키 연산에 대해서는 메모리 사용과 연산 속도 최적화를 각각 목표로 하는 두 가지 방식을 제안하였다. 메모리 최적화 방안은 라운드키를 8비트 단위로 저장하고 연산 시 32비트로 확장하여 메모리 사용량을 줄이는 방식이며 속도 최적화 방안은 사전 키 스케줄 과정에서 라운드키를 32비트로 확장하여 저장함으로써 연산 속도는 향상되지만 메모리 소모가 증가하는 방식이다.

Ⅲ. 제안 기법

CTR 모드는 블록 암호를 스트림 암호처럼 활용할 수 있도록 설계된 방식으로 각 블록이 독립적으로 암호화되기 때문에 병렬 처리에 적합하다. 그러나 기존 방식에서는 각 스레드가 하나의 평문 블록을 개별적으로 암호화하기 때문에 스레드 간 연산이 독립적이라 하더라도 병렬화의 이점을 충분히 활용하지 못하는 문제가 발생할 수 있다.

각 평문 블록을 개별적으로 암호화할 경우 각 스레드가 서로 다른 키 스트림을 생성하고 개별적인 연산을 수행해야 한다. 이 과정에서 스레드마다 별도로 메모리에 접근해야 하므로 전체적인 메모리 접근 횟수가 증가하고 연산 속도가 저하될 수 있다. 이를 해결하기 위해 본 연구에서는 4개의 평문을 묶어 암호화를 수행함으로써 병렬화 성능을 극대화하는 방식을 제안한다.

1. 공유 메모리 활용

CUDA의 공유 메모리는 같은 블록 내의 모든 스레드가 접근할 수 있으며 글로벌 메모리보다 빠른 속도를 제공한다. 따라서 동일한 데이터를 여러 스레드가 반복적으로 사용해야 하는 경우 글로벌 메모리가 아닌 공유 메모리에 저장하여 활용하는 것이 더 효율적이다,

기존의 GPU 기반 PIPO 블록 암호 최적화 연구에서는 자주 사용되는 라운드키 값을 공유 메모리에 저장하여 활용함으로써 글로벌 메모리 접근 횟수를 줄이고 연산 속도를 향상시켰다[14]. 그러나 본 연구에서는 공유 메모리에 라운드키가 아닌 마스터키를 저장하고 필요할 때 Packing을 하는 방식을 적용하였다. 이를 통해 연산의 효율성을 높여 성능을 최적화하였다.

2. Byte Permutation

CUDA의 __byte_perm 함수는 바이트 단위의 데이터를 재배열하여 32비트 정수를 생성하는 C 내장 함수이다. 이 함수는 단일 명령어로 실행되므로 기존의 여러 연산이 필요한 방식보다 더 빠르고 효율적인 데이터 변환을 수행할 수 있다.

특히 __byte_perm은 레지스터 내에서 직접 연산을 수행하므로 불필요한 메모리 접근을 줄이고 병렬 연산에 적합한 형태로 데이터 처리가 가능하다. 이를 활용하면 __byte_perm 함수를 통해 데이터 Packing 및 Unpacking 작업을 최적화할 수 있다.

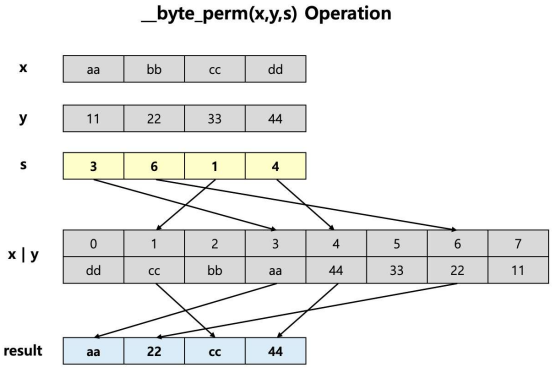

그림 3. Byte Permutation Operation

__byte_perm 함수는 두 개의 32비트 정수 a와 b에서 특정 바이트를 선택하여 새로운 32비트 정수를 생성한다. 특정 바이트를 선택하는 기준은 s가 된다. s는 바이트 선택을 위한 16비트 정수이며 각 4비트는 특정 바이트의 인덱스를 나타낸다. 각 4비트 값은 0~7 사이의 숫자로 0~3은 a의 최하위 바이트부터 최상위 바이트까지를 나타내고 4~7은 b의 최하위 바이트부터 최상위 바이트까지를 나타낸다. s의 최상위 4비트부터 차례로 a와 b의 바이트 값을 가져와 4개의 바이트를 조합하여 새로운 32비트 정수를 생성한다.

가. 평문 Packing 및 Unpacking

4개의 평문을 하나의 연산에서 동시에 처리하기 위해서는 각 평문의 데이터를 메모리에서 불러와 정렬하는 과정이 필요하다. 그러나 일반적인 방식으로 메모리를 복사하여 재배열하면 메모리 접근 횟수가 증가하고 연산 속도가 저하될

수 있다. 이를 해결하기 위해 본 연구에서는 __byte_perm 함수를 이용한 Packing 기법을 적용하였다. Packing 과정에서는 4개의 64비트 평문 블록을 32비트 단위로 변환한다. 이후 byte_perm 함수를 이용해 각 블록의 바이트를 특정 위치로 이동시켜 연산에 적합한 데이터 형태로 변환한다. 바이트의 위치를 조정하기 위해 PERM_값을 사용하며 이 값은 16비트로 구성된다. 이러한 방식은 불필요한 메모리 접근을 줄이고 연산 속도를 최적화하는 효과를 제공한다.

Packing 전 각 평문은 8개의 바이트로 구성되며 하나의 블록으로 저장된다. 이 상태에서 Sbox, Pbox, KeyAddition 연산을 수행할 경우 각 평문블록이 개별적으로 처리되므로 병렬 연산의 효율이 떨어질 수 있다. 따라서 4개의 평문에 대해 해당 연산이 한 번에 실행될 수 있도록 Packing 과정이 적용된다.

```
__device__ void PERM_IN(u32* input, u32* output_4) {
u32 tmp[8];
tmp[0] = __byte_perm(input[2], input[0], PERM_1);
tmp[1] = __byte_perm(input[2], input[0], PERM_2);
...
tmp[6] = __byte_perm(input[7], input[5], PERM_1);
tmp[7] = __byte_perm(input[7], input[5], PERM_2);

output_4[0] = __byte_perm(tmp[4], tmp[0], PERM_3);
output_4[1] = __byte_perm(tmp[4], tmp[0], PERM_4);
...
output_4[6] = __byte_perm(tmp[7], tmp[3], PERM_3);
output_4[7] = __byte_perm(tmp[7], tmp[3], PERM_4);
}
```

그림 4. PT Packing Process

PT0	A7	A6	A5	A4	A3	A2	A1	A0
PT1	B7	B6	B5	B4	B3	B2	B1	B0
PT2	C7	C6	C5	C4	C3	C2	C1	C0
PT3	D7	D6	D5	D4	D3	D2	D1	D0

Before Packing PT

PT0	D0	C0	B0	A0
PT1	D1	C1	B1	A1
PT2	D2	C2	B2	A2
PT3	D3	C3	B3	A3
PT4	D4	C4	B4	A4
PT5	D5	C5	B5	A5
PT6	D6	C6	B6	A6
PT7	D7	C7	B7	A7

After Packing PT

그림 5. PT State Before and After Packing

Packing 이후 기존 4개의 블록이 8개의 블록으로 재구성되며 평문의 상태는 그림5와 같다. 각 평문의 동일한 인덱스를 가진 바이트들이 하나의 새로운 평문 블록으로 정렬된다. 이러한 변환을 통해 Sbox, Pbox, KeyAddition 연산을 모든 평문에 대해 동시에 수행할 수 있는 환경이 조성되며 병렬 연산의 성능을 극대화할 수 있다.

암호화가 완료된 후에는 다시 각 평문을 원래 형태로 변환해야 한다. 이를 위해 __byte_perm 함수를 사용하여 4개의 평문을 각 블록 단위로 다시 정렬하여 개별적인 암호문 블록을 생성한다. Unpacking 과정은 Packing 과정의 역연산을 수행하여 원래 데이터 구조를 복원하는 것이 중요하다.

Unpacking을 수행하지 않으면 암호화된 데이터가 개별 블록과 동일한 형식을 유지하지 않으며 복호화 및 추가적인 연산이 어려워질 수 있다. 따라서 Packing 시 적용된 PERM_IN 연산의 역연산을 수행하여 개별 블록을 복원해야 한다. 이 과정에서는 PERM_IN 연산의 역연산인 PERM_OUT 연산을 사용했으며 PERM_값 대

신 PERM_INV_값을 사용하여 역변환을 수행한다.

```
__device__ void PERM_OUT(u32* input, u32* output_4) {
u32 tmp[8];
tmp[0] = __byte_perm(input[0], input[1], PERM_INV_1);
tmp[1] = __byte_perm(input[0], input[1], PERM_INV_2);
...
tmp[6] = __byte_perm(input[6], input[7], PERM_INV_1);
tmp[7] = __byte_perm(input[6], input[7], PERM_INV_2);

output_4[0] = __byte_perm(tmp[2], tmp[0], PERM_INV_3);
output_4[1] = __byte_perm(tmp[6], tmp[4], PERM_INV_3);
...
output_4[6] = __byte_perm(tmp[3], tmp[1], PERM_INV_4);
output_4[7] = __byte_perm(tmp[7], tmp[5], PERM_INV_4);
}
```

그림 6. PT Unpacking Process

나. 마스터키 Packing

본 연구에서는 4개의 평문을 묶어 Packing하여 병렬로 처리하는 최적화 기법을 적용하였다. 이 방식을 사용하면 데이터 배치가 기존과 달라지므로 마스터키 역시 동일한 방식으로 Packing하여 변형해야 한다. 이를 위해 __byte_perm 함수를 활용하여 마스터키의 바이트 순서를 재배열하고 공유 메모리에 저장된 마스터키를 기반으로 변형된 라운드키를 생성하였다. 라운드키 생성은 홀수 라운드와 짝수 라운드를 나눠 생성하였으며 Packing 후 라운드키의 상태는 그림 8과 같다.

```
__device__ void PERM_COPY(u32* input, u32* output_4) {
output_4[0] = byte_perm(input[0],input[0],PERM_COPY_0);
output_4[1] = byte_perm(input[0],input[0],PERM_COPY_1);
output_4[2] = byte_perm(input[0],input[0],PERM_COPY_2);
output_4[3] = byte_perm(input[0],input[0],PERM_COPY_3);
output_4[4] = byte_perm(input[1],input[1],PERM_COPY_0);
output_4[5] = byte_perm(input[1],input[1],PERM_COPY_1);
output_4[6] = byte_perm(input[1],input[1],PERM_COPY_2);
output_4[7] = byte_perm(input[1],input[1],PERM_COPY_3);
}
```

그림 7. MK Packing Process

MK0	K7	K6	K5	K4	K3	K2	K1	K0
MK1	K15	K14	K13	K12	K11	K10	K9	K8

Before Packing MK

RK_EVEN0	K0	K0	K0	K0	RK_ODD0	K8	K8	K8	K8
RK_EVEN1	K1	K1	K1	K1	RK_ODD1	K9	K9	K9	K9
RK_EVEN2	K2	K2	K2	K2	RK_ODD2	K10	K10	K10	K10
RK_EVEN3	K3	K3	K3	K3	RK_ODD3	K11	K11	K11	K11
RK_EVEN4	K4	K4	K4	K4	RK_ODD4	K12	K12	K12	K12
RK_EVEN5	K5	K5	K5	K5	RK_ODD5	K13	K13	K13	K13
RK_EVEN6	K6	K6	K6	K6	RK_ODD6	K14	K14	K14	K14
RK_EVEN7	K7	K7	K7	K7	RK_ODD7	K15	K15	K15	K15

After Packing MK

그림 8. MK State Before and After Packing

마스터키 Packing을 수행하면 공유 메모리에 있는 마스터키를 이용해 라운드키를 생성한다. 라운드키는 미리 생성하지 않고 매 라운드마다 필요한 라운드키를 일부 계산하여 사용한다. 해당 방법을 통해 라운드 키를 저장하지 않아 메모리를 절약할 수 있다. 생성된 라운드키는 평문 Packing의 결과와 동일한 데이터 배열을 유지하여 적용되므로 데이터 정렬을 유지하고 암호화 연산의 일관성을 보장한다. 이러한 방식은 전역 메모리 접근을 최소화하여 GPU 성능을 최적화하는데 기여한다.

3. 비트 연산

PIPO 블록 암호는 S-Layer와 R-Layer 연산이 모두 8비트 단위로 수행된다. 본 연구에서는 S-Layer와 R-Layer를 비트 슬라이싱 방식으로 변환하여 4개의 입력 블록을 동시에 처리할 수 있도록 구현하였다. S-Layer에서 4개의 입력 블록을 비트 슬라이싱 하여 동일한 인덱스의 바이트를 32비트 단위로 저장함으로써 하나의 연산으로 4개 블록에 대한 처리를 동시에 수행할 수 있도록 하였다. R-Layer 또한 동일한 방식으로 비트 슬라이싱을 적용하되 로테이션 시에는 마스킹 기법을 활용하여 데이터 간 간섭을 방지하였다. 이러한 방식을 통해 4개의 입력 블록을 병렬로 처리함으로써 연산 효율을 향상시켰으며 전체 구조는 그림 9와 같다.

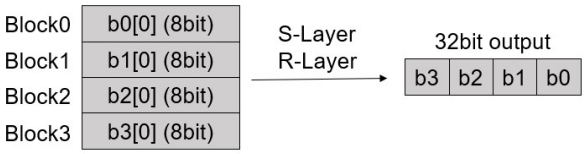

그림 9. S-Layer and R-Layer processing

Ⅳ. 성능 평가

본 연구에서는 RTX 3060 Mobile과 RTX 4080 두 가지 GPU 환경에서 PIPO 병렬 구현의 성능을 측정하였다. 첫 번째 환경인 RTX 3060 Mobile은 낮은 소비 전력과 준수한 성능을 제공하지만 전력 제한에 따라 성능 차이가 크며 데스크톱 버전에 비해 연산 성능이 제한된다. 두 번째 환경인 RTX 4080은 딥러닝, 암호화 연산 등 고성능 연산 작업에 최적화 된 GPU로 더 높은 연산 처리량과 메모리 대역폭을 활용하여 병렬 암호화 성능을 향상시킬 수 있다.

표 1. Comparison of PIPO Performance by GPU

MODE	Environment	PIPO-128 (Tbps)	PIPO-128 (CPB)	PIPO-256 (Tbps)
CTR	RTX 3060 Mobile	1.18	23.73	0.89
	RTX 4080	4.66	6.01	3.56
ECB	RTX 2080Ti[15]	1.11	25.23	-
	GTX 1650[15]	0.27	103.32	-
	Intel Core i9[Ref C] (32 Block Encryption)	1.07	26.05	-
	Intel Core i9[Ref C] (64 Block Encryption)	1.11	25.22	-

표 1은 CPU 및 다양한 GPU 환경에서 PIPO 블록 암호의 연산 성능을 비교한 결과를 보여준다. CPU 환경에서는 32개 및 64개의 블록을 처리하였으며 이에 따라 각각 1.07 Tbps 및 1.11 Tbps의 처리량을 기록하였다. 반면 본 연구에서 제안한 GPU 기반 구현은 4개의 블록만을 병렬 처리했음에도 불구하고 환경에 따라 1.18 Tbps 및 4.66 Tbps의 높은 처리 성능을 달성하였다. 이는 병렬 처리 블록 수가 CPU 대비 현저히 적음에도 불구하고 GPU의 병렬성 및 메모리 대역폭을 효과적으로 활요함으로써 CPU 구현보다도 뛰어난 처리량을 달성하였음을 입증한다. GPU 간의 성능을 비교해보기 위해서는 기존 연구의 결과를 이용하였다[15]. 기존 연구는 CTR 모드 구현이 아니라서 직접적인 비교에는 한계가 존재할 수 있다. 기존 연구에서 RTX 2080Ti는 PIPO-128 구현에서 1.11 Tbps의 성능을 기록하였다. 반면 본 연구의 CTR 모드 구현에서 RTX 4080은 가장 높은 성능인 4.66 Tbps으로 이는 기존 연구 대비 약 4.2배 더 높은 성능을 보여주는 것으로 확인했다. 또한 RTX 3060 Mobile은 1.18 Tbps(PIPO-128), 0.89 Tbps(PIPO-256)의 성능을 보이며 기존 연구 대비 약 1.06배 더높은 성능을 기록하였다. 이는 RTX 2080Ti가 RTX 3060 Mobile보다 기본적으로 연산 성능이 뛰어난 GPU임에도 불구하고 본 연구에서 최적화 기법을 적용함으로써 더 높은 성능을 달성할 수 있었음을 의미한다.

연산 효율성을 비교하기 위해 CPB 지표도 같이 분석하였다. 처리량과 함께 분석함으로써 단순 처리 속도 뿐만 아니라 자원 활용 효율성까지 평가하였다. 본 연구의 GPU 기반 구현은 RTX 4080에서 4.66 Tbps의 처리 속도와 6.01의 낮은 CPB를 동시에 기록하여, 높은 처리량과 효율성을 동시에 달성하였음을 확인할 수 있다. 반면 CPU 구현은 1.11 Tbps의 처리 속도를 보였지만 CPB는 25.23으로 GPU 구현 대비 연산 효율이 낮음을 보여준다.

이러한 결과는 CTR 모드의 병렬 연산 최적화 효과와 GPU의 아키텍처 차이로 인해 발생한 성능 차이로 보인다. 특히 RTX 4080과 RTX 3060 Mobile 간의 성능 차이를 고려했을 때 CUDA 코어 개수, 메모리 대역폭, 연산 구조 등이 암호화 연산 성능에 미치는 영향이 크다는 점을 확인할 수 있다.

Ⅴ. 결론

본 논문에서는 GPU 상에서 PIPO 블록 암호의 CTR 모드를 최적화하는 방법을 제안하였다. 기존의 암호화 방식과 비교하여 4개의 평문을 병렬 처리하는 방식을 적용함으로써 전반적인 연산 속도를 향상시킬 수 있었다. 이를 위해 Packing 및 Unpacking 기법을 도입하여 데이터 재배열을 최적화하였으며 공유 메모리에 저장된 마스터키를 사용하여 글로벌 메모리 접근을 최소화함으로써 성능을 향상시켰다. 성능 평가 결과, 본 연구에서 제안한 최적화 기법을 적용한 구현이 기존 연구 대비 효율적임을 확인할 수 있었다. 이는 GPU 기반의 블록 암호 최적화에 있어 Packing 및 Unpacking 기법과 공유 메모리를 활용이 효과적이었음을 나타낸다.

본 연구에서 제안한 최적화 기법은 PIPO뿐만 아니라 다른 다양한 블록암호에도 적용 가능할 것으로 보인다. 향후 연구에서는 다양한 블록 크기 및 키 크기에 따른 최적화 방식의 성능 차이를 분석하고 최적화 기법을 적용하는 방안을 제시할 수 있을 것으로 기대한다.

REFERENCES

[1] Dar, Aaqib Bashir, Mashhood Jeelani Lone, and Nuzhat Hussain. "Revisiting lightweight block ciphers: review, taxonomy and future directions," *Computer Science Review, Forthcoming,* 2021.

[2] An, S.; Seo, S.C. Highly Efficient Implementation of Block Ciphers on Graphic Processing Units for Massively Large Data. Appl. Sci. 2020, 10, 3711. https://doi.org/10.3390/app10113711

[3] Iwai, Keisuke, Takakazu Kurokawa, and Naoki Nisikawa. "AES encryption implementation on CUDA GPU and its analysis," *2010 First international conference on networking and computing. IEEE,* 2010.

[4] Kim, Hangi, et al. "PIPO: A lightweight block cipher with efficient higher-order masking software implementations," Information Security and Cryptology–ICISC 2020: 23rd International Conference, Seoul, South Korea, December 2–4, 2020, Proceedings 23. Springer International Publishing, 2021.

[5] Kim, Sunyeop, et al. "Integral cryptanalysis of lightweight block cipher PIPO," *IEEE Access,* vol. 10 pp. 110195-110204, 2022.

[6] Kim, Hangi, et al. "A new method for designing lightweight S-boxes with high differential and linear branch numbers, and its application," *IEEE Access,* vol. 9, pp. 150592-150607, 2021.

[7] N. Mouha and M. Dworkin, "Report on the Block Cipher Modes of Operation in the NIST SP 800-38 Series," Tech. Rep., NIST IR 8459, 2023.

[8] C. Tezcan, "Optimization of advanced encryption standard on graphics processing units," *IEEE Access,* vol. 9, pp. 67315–67326, 2021.

[9] M. Khairy, A.G. Wassal, and M. Zahran, "A survey of architectural approaches for improving GPGPU performance, programmability and heterogeneity," *Journal of Parallel and Distributed Computing,* vol. 127, pp. 65–88, 2019.

[10] S. W. Eum, H. J. Kim, H. D. Kwon, M. J. Sim, G. J. Song, and H. J. Seo, "Parallel Implementations of ARIA on ARM Processors and Graphics Processing Unit," *Applied Sciences,* vol. 12, no. 23, p. 12246, 2022.

[11] S. W. Eum, H. J. Kim, M. H. Song, and H. J. Seo, "Optimized Implementation of Argon2 Utilizing the Graphics Processing Unit," *Applied Sciences,* vol. 13, no. 16, p. 9295, 2023.

[12] O. Hajihassani, S.K. Monfared, S.H. Khasteh, and S. Gorgin, "Fast AES implementation: A high-throughput bitsliced approach," *IEEE Transactions on Parallel and Distributed Systems,* vol. 30, no. 10, pp. 2211–2222, 2019.

[13] Kim, Hyun-Jun, Si-Woo Eum, and Hwa-Jeong Seo. "Optimal Implementation of Lightweight Block Cipher PIPO on CUDA GPGPU," *Journal of the Korea Institute of Information Security & Cryptology,* vol. 32, no. 6, pp. 1035-1043, 2022.

[14] Choi, Hojin, and Seog Chung Seo. "Efficient parallel implementations of PIPO block cipher on CPU and GPU," *IEEE Access,* vol. 10, pp. 85995-86007, 2022.

[15] Eum, S. W., Jang, K. B., Song, G. J., Lee, M. W., Seo, H. J. "Optimized Implementation of PIPO Lightweight Block Cipher on 32-bit RISC-V Processor," *KIPS Transactions on Computer and Communication Systems,* vol. 11, no. 6, pp. 167-174, 2022.

저 자 소 개

송민호(준회원)

2023년 2월: 한성대학교 IT 융합공학과 학사 졸업.
2025년 2월: 한성대학교 IT 융합보안학과 석사 졸업.
2025년 3월: 한성대학교 IT 정보컴퓨터공학과 박사 과정.
<주관심분야 : 정보 보호, 정보 보안, 암호 구현>

엄시우(준회원)

2021년 2월: 한성대학교 IT 융합공학과 학사 졸업.
2023년 2월: 한성대학교 IT 융합공학과 석사 졸업.
2023년 3월: 한성대학교 IT 정보컴퓨터공학과 박사 과정.
<주관심분야 : 정보 보호, 정보 보안, 암호 구현>

김상원(준회원)

2023년 2월: 한성대학교 IT 융합공학과 학사 졸업.
2023년 9월: 한성대학교 IT 융합보안학과 석사 과정.
<주관심분야 : 정보 보호, 암호 구현>

서화정(정회원)

2010년 2월: 부산대학교 컴퓨터공학과 학사 졸업.
2002년 2월: 부산대학교 컴퓨터공학과 석사 졸업.
2016년 2월: 부산대학교 컴퓨터공학과 박사 졸업.
2017년 3월: 싱가포르 과학기술청
2023년 2월: 한성대학교 IT융합공학부 조교수
2023년 3월: 한성대학교 융합보안학과 부교수
<주관심분야 : 정보 보호, 암호 구현>