Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.2, Apr. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.2.235

PIM을 활용한 ORAM 가속화 연구*

신 수 환,[1†] 이 호 준[2‡]
[1,2]성균관대학교 (대학원생, 교수)

Accelerating ORAM with PIM*

Suhwan Shin,[1†] Hojoon Lee[2‡]
[1,2]Sungkyunkwan University (Graduate student, Professor)

요 약

ORAM(Oblivious RAM)은 사용자가 믿을 수 없는 서버, 혹은 하드웨어를 이용할 때 해당 기기에서 발생할 수 있는 부채널 공격을 방어할 수 있는 알고리즘이다. ORAM은 데이터 접근 패턴을 감추어 해당 접근 패턴을 통해 정보가 유실되는 것을 방어하게 된다. 그러나, ORAM은 하드웨어의 보안을 강화하나 그로 얻는 이점에 비해 훨씬 큰 처리 속도 감소를 유발하는 단점이 존재하여 현재까지 실용화 되지 못하였다. 본 논문에서는 새롭게 개발되고 있는 하드웨어인 PIM(Process In Memory)를 활용하여 ORAM을 가속화하여 실용적으로 활용할 수 있는 방안을 모색해 보고자 한다.

ABSTRACT

ORAM(Oblivious RAM) is an algorithm that defends side channel attacks when the user uses an untrusted server or hardware. ORAM defends against leaks of information by hiding data access patterns. However, ORAM is not in practical use because as ORAM reinforces hardware security, it also has a severe disadvantage in processing speed. In this paper, we suggest using newly introduced hardware, PIM (Process In Memory), to accelerate ORAM and use it practically.

Keywords: ORAM, PIM, Hardware Security

I. 서 론

최근 많은 개인들과 기업들은 클라우드 서비스[1]를 많이 사용하고 있다. 이러한 클라우드 서비스는 많은 GPU 등을 활용할 수 있게 하여 개인 혹은 기업이 부족한 계산 능력을 채우는데 사용할 수 있다.

그러나, 이러한 클라우드는 신뢰하지 못하는 환경일 경우가 많다. 클라우드 관리자는 보편적으로 클라우드에서 일어나는 모든 행동에 대해 관측할 수 있고, 설사 프로그램 등이 암호화되어 진행된다고 하더라도 보편적인 하드웨어를 사용하였을 경우 데이터 접근 패턴에 대해 알아내어 프로그램의 중요한 정보를 빼낼 수 있다.

이러한 문제점을 해결하기 위해, ORAM (Oblivious RAM) 알고리즘[2]-[4]이 개발되었다. ORAM은 사용자가 믿을 수 없는 서버, 혹은 클라우드를 이용할 때, 서버에서 발생할 수 있는 부채널 공격을 방어하는 알고리즘이다. 그러나 이러한 알고리즘은 치명적인 속도 저하를 불러일으켜 실용화되지 못하고 있다.

본 논문에서는 ORAM을 실용화할 수 있는 방법으로 PIM (Process In Memory)[5] 사용을 제

Received(02. 23. 2023), Modified(03. 16. 2023), Accepted(03. 21. 2023)
* 이 연구는 2023년도 정부의 재원으로 한국연구재단(교육부) 기초연구사업 (NRF-2022R1C1C1010494), 정보통신기획평가원(융합보안핵심인재양성 (No. 2019-0-01343), 국제공동연구사업 (No. 2020-0-00666))의 지원을 받아 수행된 연구임.
† 주저자, putt6603@skku.edu
‡ 교신저자, hjlee228@gmail.com(Corresponding author)

시한다. 최근 새롭게 개발되고 있는 PIM 하드웨어는 메모리에서 연산을 하는 방식을 가진다. 따라서, PIM 하드웨어는 단순연산을 하는 알고리즘을 처리하기에 이상적인 환경이다. 그러나 현재 상용화된 PIM의 경우 보안을 막기 위한 암호화 모델 등이 존재하지 않아 한계점이 존재한다.

본 논문은 따라서 ORAM을 탑재할 수 있는 PIM의 필요한 개선 사항을 탐구하고, 해당 개선 사항이 반영되었을 때의 ORAM을 탑재한 PIM의 성능을 분석하여 실용성을 논하고자 한다. 본 논문의 실험 결과에 따르면 기존 CPU 환경에서의 ORAM보다 PIM을 활용한 ORAM은 일부 느리게 측정되었으나, 복잡한 컴퓨팅 환경 하에서는 ORAM보다 빠르게 동작할 수 있음을 확인하였다.

II. 선행 연구

2.1 하드웨어 부채널 방어

하드웨어 부채널 공격에 방어를 하는 연구는 선행적으로 많이 진행되었다. 그 중에서 특출나게 많이 연구된 분야는 Intel SGX의 부채널 공격[7]을 방어할 수 있는 방법들로, 여러 논문에서 제시되었다. 가장 먼저, ZeroTrace[8]의 경우, Path ORAM과 Circuit ORAM 두 가지 알고리즘을 활용하여 부채널 공격을 방지하고 망각성 자료구조를 제공하고자 하였다. TrustORE[9]의 경우 SGX의 취약점을 ORAM과 FPGA를 사용하여 해결하고자 하였다. OBFUSCRURO[10]는 2개의 ORAM 트리를 사용하여 SGX 외부에서도 신뢰할 수 있는 실행 환경을 제공한다.

또한, 새로운 하드웨어를 활용하여 해당 부채널 공격을 방어하고자 하는 연구들도 존재한다. Secure DIMM[11]은 기존 ORAM들이 높은 오버헤드를 가지고 있는 점을 해결하기 위해 새로운 하드웨어인 SDIMM을 활용하는 논문이다. Invisimem[12]은 새롭게 개발되고 있는 하드웨어 중 하나인 스마트 메모리를 활용하여 메모리 버스 부채널 공격을 방어하는 기법을 소개하였다.

2.2 PATH ORAM

ORAM은 사용자가 믿을 수 없는 서버를 이용할 때, 서버에서 발생할 수 있는 부채널 공격 등을 막을 수 있도록 하는 알고리즘이다. 서버에서 암호화 과정을 사용하면 파일에 대한 기밀성이 지켜지지만 정보 접근 패턴은 파악될 수 있어 부채널 공격에 취약점이 발생한다. ORAM 알고리즘은 해당 접근 패턴을 드러내지 않게 숨김으로써 해당 공격으로부터 방어하게 된다.

PATH ORAM[6]은 현재까지 알려진 가장 단순한 ORAM 알고리즘이다. PATH ORAM은 크게 Stash와 정보를 저장하는 이진 트리의 두 부분으로 구성된다. Stash는 일반적으로 사용자가 가지고 있으며, 정보를 임시로 저장하는 역할을 한다. 이에 반해, 정보를 저장하는 이진 트리는 서버에 존재하게 된다. 사용자가 이진 트리에 정보 저장을 원하게 되면, 각 정보가 담긴 블록은 균일, 무작위하게 이진 트리의 각 리프 노드에 저장된다. 각 블록의 주소는 포지션 맵에 저장되며, 이는 신뢰할 수 있는 공간에 저장된다.

그림 1은 PATH ORAM의 Access 알고리즘을 보여준다. PATH ORAM에서 서버가 사용자에게 정보 접근 요청을 받게 되면 그림 1과 같은 알고리즘을 거치게 된다. 먼저, 접근 요청이 들어온 블록의 ID를 바탕으로 포지션 맵에서 해당 정보의 주소를 찾아 저장하게 된다. 이후, 해당 블록의 주소를 균일 무작위하게 재선정하고, 포지션 맵을 업데이트한다. 그 다음, 미리 저장한 주소를 바탕으로 ORAM 트리에서 블록의 정보를 받아 Stash에 저장하게 된다. 만약 사용자의 요청이 쓰기 요청이었을 경우, 이 과정에서 Stash에 있는 Data를 교체하게 된다. 마지막으로, 재선정된 주소를 바탕으로 ORAM 트리를 업데이트하게 됨으로써 접근 과정이 끝나게 된다.

본 논문에서는 PATH ORAM을 기본 알고리즘

Algorithm 1 PathORAM [23]'s Access(op, a, data*) :

$x \leftarrow position[a]$
$position[a] \leftarrow UniformRandom(0 \dots 2^L - 1)$
for $l \in \{0, 1, \dots, L\}$ **do**
  $S \leftarrow S \cup ReadBucket(P(x, l))$
end for
$data \leftarrow$ Read block a from S
if $op = write$ **then**
  $S \leftarrow (S - \{(a, data)\}) \cup \{(a, data^*)\}$
end if
for $l \in \{L, L-1, \dots, 0\}$ **do**
  $S' \leftarrow \{(a', data') \in S : P(x, l) = P(position[a'], l)\}$
  $S' \leftarrow$ Select $min(|S'|, Z)$ blocks from S'
  $S \leftarrow S - S'$
  $WriteBucket(P(x, l), S')$
end for

return $data$

Fig. 1. Path ORAM Access Alogirithm[6]

으로 PIM에 올라갈 ORAM을 제작하였다. PATH ORAM은 간단한 알고리즘을 가지고 있어, CPU에서 정보를 분할해주는 과정을 추가해주는 것으로 디자인 하였다.

2.3 PIM

PIM은 메모리에서 CPU에서만 연산을 수행하던 기존 Von Neumann 아키텍처와는 달리, 메모리에서도 연산 기능을 수행하도록 하는 새로운 하드웨어이다. 기존 Von Neumann 아키텍처의 경우, 메모리는 정보의 저장만을 담당했으나, CPU의 성능이 좋아짐에 따라 Von Neumann 병목현상이 일어나 새로운 연산 시스템의 필요성이 대두되었다. 이러한 필요성으로 인해 PIM이 개발되었다. PIM은 국내외의 다양한 기업들에서 현재 개발 및 상용화를 연구중에 있다. [13]-[15]

본 논문에서는 UPMEM사에서 개발한 PIM(이하 UPMEM)[10]을 바탕으로 연구를 진행한다. 그림 1은 UPMEM의 한 DPU 구조를 보여준다. UPMEM은 랭크 단위로 구분되어 있으며, 한 랭크 내부에는 64개의 DPU가 있고, 이러한 랭크가 2개 모여 만들어진 8GB DIMM 내부에는 128개의 DPU가 존재한다. 실험 환경에서 사용한 UPMEM은 총 32개의 랭크와 4096개의 DPU로 구성되어 있다. 각 PIM은 DPU (Data Processor Unit), IRAM, WRAM, DMA 엔진으로 구성되어 있다. DMA 엔진은 각 IRAM, WRAM, MRAM 간의 통신을 관리하는 역할을 하게 되며, WRAM은 실행 시에 DPU가 스크래치 패드로 활용하게 된다.

UPMEM의 성능을 측정한 선행 연구[16]에 따르면, UPMEM은 정보 복사에 대해서는 기존의 CPU 연산 체계에 비해 빠른 성능을 보였다. 그러나 AES 암호화 및 복호화 같은 복잡한 연산이 들어가는 과제의 경우 기존 CPU 연산에 비해 매우 느린 연산 속도를 보여주었다. PIM을 ORAM 연산을 위해 활용할 경우, 복잡한 연산 대신 정보의 이동이 주로 일어나기 때문에 기존 ORAM보다 빠른 처리 속도를 가질 것으로 기대하였다.

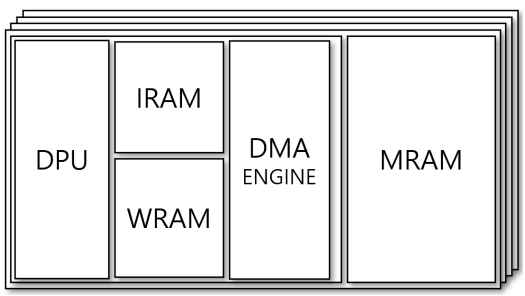

Fig. 2. The structure of UPMEM

III. 연구방법

3.1 공격자 모델과 방어 모델 분석

본 논문에서는 공격자 모델을 다음과 같이 정의하였다. 공격자는 여러 가지 강력한 공격을 할 수 있는 것으로 알려진 물리적 하드웨어 부채널 공격[17]과 같은 서버를 이용하여 소프트웨어적인 공격을 가할 수 있다. 해당 공격을 통해 공격자는 메모리 접근 패턴을 획득할 수 있으며, 해당 정보를 이용하여 사용자가 이용한 정보 등의 치명적인 정보를 획득할 수 있다. 또한 CPU와 메모리 사이를 연결하는 버스에 타이밍 공격을 하여 접근하는 메모리를 알아내거나, 버스 탈취 공격을 하여 명령어를 가로채려는 시도를 할 수 있다. 마지막으로, 콜드 부트 어택 혹은 소프트웨어적인 방법으로 메모리 자체의 정보를 알아내려는 공격을 시도할 수 있다.

이러한 공격들을 방어하기 위해 몇 가지를 가정한다. 먼저, CPU와 PIM 코어 프로세서는 신뢰실행 환경 내부에 있어 신뢰할 수 있다고 가정한다. 또한, PIM 내부의 WRAM과 IRAM은 기밀성과 무결성이 보장되며 신뢰할 수 있다고 가정한다. MRAM은 기밀성과 무결성은 보장되지만 공격자에게 소프트웨어 등으로 정보가 탈취될 수 있어 신뢰할 수 없다고 가정한다. WRAM과 IRAM의 경우 On-Chip 메모리로써 콜드 부트 공격 등으로부터 안전하다는 연구 결과[19]가 존재하여 정보가 탈취될 가능성이 적다. 이에 반해 MRAM을 신뢰할 수 없는 것은 시스템 소프트웨어를 통해 접근 가능하기 때문이다.

ORAM에서 보안을 유지하기 위해서는 포지션 맵이 유출되어서는 안된다. 그러나, 계속해서 균일 무작위하게 정보가 다시 저장되는 ORAM 트리의 경우 비밀 정보가 아니다. 따라서 포지션 맵을 공격자가 정보를 얻어낼 수 없는 WRAM에 위치하게 되면 공격자로의 유출을 방지할 수 있다.

또한, PATH-ORAM과는 다르게 만약 우리의 설계에서 각 ORAM 트리에 한 개의 정보를 저장하게 된다면 공격자는 어느 DPU의 ORAM에 접근했

는지를 파악해 정보를 얻어낼 수 있는 위험이 있다. 이러한 문제점은 정보를 나누어 여러 ORAM에 저장하는 것으로 해결하였다. 여러 ORAM에 나누어 저장하게 되면 한 특정 정보를 저장하거나 수정할 때 모든 ORAM에 접근이 이루어지게 되므로 공격자가 어떠한 정보에 접근하였는지 판단을 할 수 없게 된다.

마지막으로 MRAM과 버스에서의 취약점을 해결하기 위해 CPU에서 DPU로 데이터를 보낼 때와 MRAM에 데이터를 저장할 때 암호화가 필요하다. 그러나, 현재 PIM의 경우 암호화와 비암호화를 위한 복잡한 연산이 오버헤드가 커 실용적이지 않다. 따라서, 본 논문에서는 해당 암호화 모듈이 PIM 내부에 존재한다고 가정하여 성능 분석을 진행하였다.

3.2 O-PIM 설계

O-PIM은 PATH ORAM의 기본 알고리즘을 PIM 상에 구현, 동작하는 것을 목표로 하였다. PATH ORAM의 경우 현재까지 개발된 ORAM 알고리즘 중 가장 간단한 알고리즘으로, 해당 알고리즘을 사용하였을 때 가장 현실적인 설계 방안이 될 것이다.

O-PIM에서 각 부분은 다음과 같이 위치한다. CPU에는 사용자가 요청한 정보가 존재한다고 가정한다. 신뢰할 수 있는 WRAM에는 Stash와 포지션 맵, 그리고 명령어 버퍼와 데이터 버퍼를 위치하였다. 명령어 버퍼는 CPU로부터 전달받은 명령어를 저장하는 곳이다. 명령어의 종류로는 ORAM을 초기화하는 INITIALIZE(), ORAM에서 데이터를 읽어오는 READ(ID), ORAM에 데이터를 작성하거나 수정하는 WRITE(ID, DATA)가 존재한다. 데이터 버퍼는 CPU로부터 전달받은 데이터를 임시 저장하는 장소이다. 신뢰할 수 없는 MRAM에는 메인 데이터 저장소인 ORAM 트리가 위치한다. ORAM 트리는 PATH ORAM과 마찬가지로 이진 트리로 구성되며, 각 리프에 저장되는 블록들은 데이터와 ID같은 메타데이터로 구성되어 있다.

그림 3은 O-PIM 동작을 대략적으로 보여준다. 먼저, O-PIM이 포함된 시스템을 처음 시동하는 경우, CPU는 시스템에서 지시받은 개수의 DPU를 할당하여 ORAM 트리를 해당 개수만큼 만들게 된다. 이후 사용자가 ORAM 트리에 쓰기 명령어를 준 경우를 가정해 보자. ① CPU에서 받은 쓰기 명령어를 각 DPU 내부의 명령어 버퍼로 전달한다. ② CPU는 데이터를 DPU가 할당된 개수만큼 나눈다. ③ CPU에서 나눈 각각의 데이터를 각 DPU의 데이터 버퍼로 전달한다. ④ Stash로부터 ID를 바탕으로 블록의 주소값을 알아낸 다음, 해당 주소를 바탕으로 MRAM의 ORAM 트리로부터 데이터를 Stash에 저장한다. ⑤ 데이터 버퍼의 데이터를 바탕으로, Stash에 저장된 블록을 업데이트 한다. 이 과정에서 동시에 DPU에서 균일 무작위로 생성한 주소값으로 추가 변경하게 된다. ⑥ Stash에서 변경받은 주소값을 바탕으로 포지션 맵을 업데이트 하게 된다. ⑦ 마지막으로, Stash에서 업데이트 받은 정보를 바탕으로 ORAM 트리를 업데이트하며 동작을 마치게 된다.

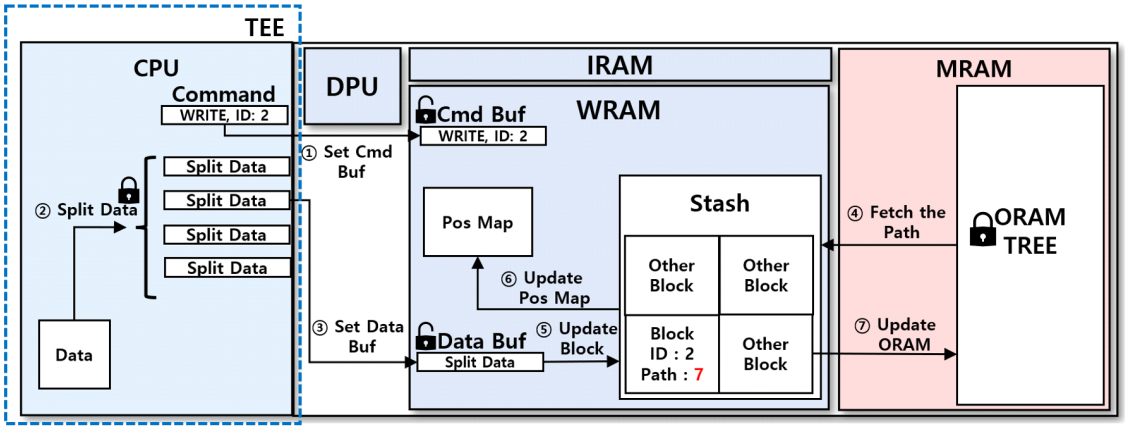

Fig. 3. O-PIM Operation Overview

3.3 실험 설계

본 실험은 UPMEM PIM 서버에 위에서 설명한 O-PIM을 C와 C++으로 구현하여 실행하는 것으로 설계하였다. 2개의 실험을 설계하여 진행하였다. 첫 번째 실험은 O-PIM의 성능을 정량적으로 측정하는 것을 목표로 하였다. 두 번째 실험은 CPU만을 활용한 PATH ORAM 알고리즘과의 비교를 통해 O-PIM이 실용성이 있는지 검증하고, 개선점을 찾고자 설계하였다.

첫 번째 실험은 O-PIM의 성능을 측정하는 실험이다. 해당 실험은 UPMEM PIM의 서버 환경에서 DPU를 8개에서 256개까지 늘려가며 할당하여 실행하는 것으로 계획하였다. 각 실험은 변인마다 10회씩 실행되었으며, O-PIM에게는 총 1000개의 데이터를 쓴 이후, 1000회 읽어오는 과제가 주어졌다. DPU를 늘렸을 때, O-PIM의 실행 속도가 실제로 가속되는지 관찰하였다.

두 번째 실험은 O-PIM과 기존의 PATH ORAM을 비교하는 실험이다. 첫 번째 실험에서 얻은 결과물과 비슷한 용량의 ORAM 트리를 가진 CPU만을 활용하여 동작하는 PATH ORAM을 UPMEM 서버상에서 실행시켜 얻은 결과물을 서로 비교 분석하였다. 또한, 실제 클라우드 서버 상황에서 많은 양의 데이터가 오고 가고 있는 상황을 가정하기 위하여 Intel Steam Benchmark를 백그라운드에 실행하여 서버를 바쁘게 한 상황도 가정하여 추가 실험을 진행하였다. 각 실험은 10회씩 진행하여 총 걸린 시간의 평균을 측정하였다.

실험에 사용된 UPMEM 서버의 성능은 표 1에 나타나 있다.

Table 1. System information on evaluation device

CPU	Intel(R) Core(TM) i9-10900K
Memory	125.6GB
Kernel	5.15.0-52-generic

IV. 연구결과

4.1 O-PIM 성능 측정

첫 번째 실험으로 UPMEM 서버에서 구현한 O-PIM을 DPU 8개부터 512개를 할당하여 성능을 측정하였다. 각 시행마다 ORAM에 총 1000개의 블록을 입력한 이후, 1000개의 블록을 다시 읽는 과제를 총 10회 수행하고, 해당 과제들의 평균 시간을 측정하였다. 그림 4는 할당된 DPU의 개수의 변화에 따른 읽기 시간 변화를 보여준다.

위 그래프에서의 DPU Execution Time은 DPU가 연산을 처리하는 시간이고, Data Transfer Time은 정보를 CPU에서 DPU로 복사, 혹은 DPU에서 CPU로 복사하는 시간을 뜻한다.

위의 결과에 따르면, DPU 개수가 증가해도 각 DPU의 Execution Time은 상대적으로 안정적임을 확인할 수 있었다. 그러나 Data Transfer Time은 DPU가 증가할수록 늘어나는 형태를 보였다. 이를 통해 알 수 있는 점은 DPU를 통해 연산 능력이 분산되기 때문에 작업 부하가 늘어나도 Execution Time은 일정하다는 것을 알 수 있었다.

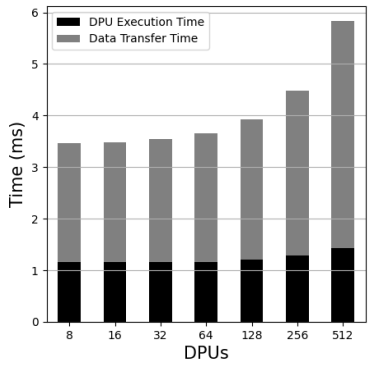

Fig. 4. O-PIM performance evaluation

4.2 O-PIM과 PATH ORAM 성능 비교

PATH ORAM을 일반적인 서버와 바쁜 서버에서 돌렸을 때, 그리고 O-PIM을 바쁜 서버에서 돌렸을 때에 대한 경우들을 실험하였다. 가장 먼저, 그림 5는 PATH ORAM을 일반적인 백그라운드가 적은 서버에서 돌렸을 때의 결과를 보여준다. x축은 ORAM의 저장 공간을 나타내고, y축은 걸린 시간을 나타내는데, 256kb가 O-PIM의 8 DPU가 할당되는 공간에 해당된다. 위 그래프와 실험 1에서의 그림 4 그래프를 비교해보면 알 수 있듯이, PATH ORAM은 O-PIM에 비해 13배 이상 빠른 동작을 보였다.

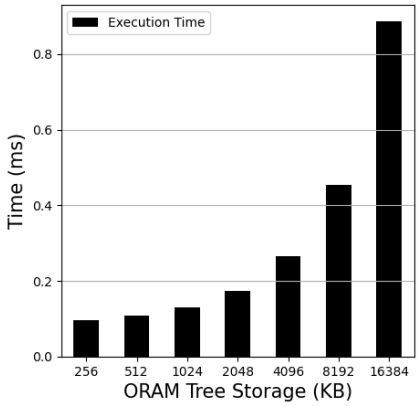

Fig. 5. PATH ORAM performance evaluation on normal server

그러나 백그라운드에 서버가 바쁜 상황을 가정하여 Stream Benchmark를 실행한 상태로 O-PIM은 512개의 DPU를 할당하고 과제를 수행하였을 때와 PATH ORAM에서 16384KB의 공간을 할당하여 과제를 수행하고 이를 비교하였을 때는 위와 상반된 결과가 나왔다. O-PIM은 연산 능력이 DPU간에 분산되어 1.22배 느려진 반면, CPU는 5.8배 가량 느려진 것으로 나타났다. 자세한 실험 결과는 표 2에 정리되어 있다.

해당 결과는 실험1에서 판단되었 듯이, DPU Execution Time은 작업 부하가 많아져도 일정한 반면, CPU Execution Time은 그에 비해 작업 부하가 많아지면 느려질 수 있기 때문이라고 생각된다. O-PIM에 더 많은 DPU가 할당되어 더 많은 용량을 감당하게 된다면, 해당 격차가 더욱 줄어들 것으로 예상된다. 본 실험에서 사용한 ORAM보다 더 큰 용량을 감당해야하고, 더욱 바쁜 서버 상황에서는 PATH ORAM보다 O-PIM이 실용적일 것이다.

Table 2. Comparison on O-PIM and PATH ORAM (unit : ms)

ORAM Type	General Situation	Busy Server
O-PIM	5.82	7.15
PATH ORAM	0.45	2.63

4.3 보안 분석

공격자 모델에서 정의한 공격을 공격자가 한다고 가정해 보자. 공격자는 물리적 하드웨어 부채널 공격을 PIM을 포함한 시스템에 가할 수 있다. 이 공격의 목적은 사용자가 어떠한 물리적 저장소에 접근하는지 판단하여, 사용자의 정보를 얻어내기 위함일 것이다. 이 경우, 공격자는 데이터 접근 패턴을 살피게 될 것이다.

본 논문에서 설계한 O-PIM을 활용한다고 가정해 보자. 공격자는 기밀성과 무결성이 보장된 WRAM 및 IRAM은 데이터 접근을 관측할 수 없어, ORAM 트리가 포함된 MRAM을 관측하게 될 것이다. 이 경우, ORAM에서 가장 중요한 정보라고 할 수 있는 포지션 맵은 WRAM에 위치하기 때문에 공격자가 관측할 수 없다. 또한, MRAM으로의 정보 접근은 ORAM 트리로부터 정보를 읽어올 때, 그리고 ORAM 트리에 정보를 쓸 때의 두 가지의 종류가 있는데, 공격자는 해당 접근을 읽을 수 있다. 그러나, ORAM 트리의 경우 데이터를 여러 랭크에 걸쳐 나누어 저장하므로, 정확히 어떠한 정보에 접근하였는지, 혹은 어떠한 정보를 업데이트하였는지는 공격자가 파악할 수 없다.

또한, 공격자가 버스에 공격을 가하더라도 모든 DPU로 데이터가 동시에 이동하므로 어떠한 데이터가 오가는지 공격자는 판단할 수 없다. 그리고 버스 탈취 공격에 대해서도 암호화된 명령어가 이동하므로 안전한다. 메모리에 소프트웨어를 활용한 공격이 가해질 경우, 버스와 MRAM 등은 암호화된 데이터가 오가므로 이 경우 역시 공격자는 정보를 얻지 못한다.

V. 결 론

본 논문에서는 PIM을 활용하여 ORAM을 가속화 하는 방안을 제시하였다. 연구 결과에서 알 수 있듯이 DPU 개수가 증가하면 속도가 증가하는 것을 확인 할 수 있었다. 그러나 PATH ORAM과 비교해 보았을 때는 O-PIM이 PATH ORAM에 비해 속도가 느린 것을 알 수 있었다. 이는 CPU 와 DPU 간 복사 속도의 한계에 의한 것으로, 해당 부분이 개선되면 PATH ORAM에 비해 빠른 속도가 나올 것으로 기대된다.

또한, 보안상 한계점도 존재한다. 소프트웨어적인 부분의 방어 및 버스 탈취 공격의 방어를 위해 암호

화 및 복호화가 필요로 하나 현재 PIM은 암호화, 복호화를 위한 복잡한 연산이 매우 느려 ORAM에 암호화 및 복호화를 추가하게 되면 속도가 굉장히 느려진다. 만약 ORAM을 PIM에서 상용화하려고 한다면 이러한 점이 개선 되어야 할 것이다.

본 연구를 진행하면서 PIM의 보안에서의 잠재능력과 한계점을 동시에 확인할 수 있었다. PIM이 하드웨어적인 몇 가지 약점을 보완하게 된다면 ORAM을 충분히 PIM에서 활용하여 실용적으로 사용하게 될 수 있을 것이다.

References

[1] Min, Ok-Gi, Kim, Hak-Yeong, and Nam, Gung-Han, "Trends in Technology of Cloud Computing," Electronics and Telecommunications Trends, 24(4) pp. 1-13, Aug. 2009.

[2] O. Goldreich, "Towards a theory of software protection and simulation by Oblivious Rams," Proceedings of the nineteenth annual ACM conference on Theory of computing, pp. 182-194, Jan. 1987.

[3] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," Journal of the ACM, vol. 43, no. 3, pp. 431 - 473, May. 1996.

[4] X. Wang, H. Chan, and E. Shi, "Circuit oram," Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 850-861, Oct. 2015.

[5] J. Draper, J. Chame, M. Hall, C. Steele, T. Barrett, J. LaCoss, J. Granacki, J. Shin, C. Chen, C. W. Kang, I. Kim, and G. Daglikoca, "The architecture of the diva processing-in-memory chip," Proceedings of the 16th international conference on Supercomputing, pp. 14-25, Jun. 2002.

[6] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, "Path Oram," Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp.1-26, Apr. 2013.

[7] Y. Zhang, M. Zhao, T. Li, and H. Han, "Survey of attacks and defenses against SGX," 2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC), pp.1492-1496, Jun. 2020.

[8] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace : Oblivious memory primitives from Intel SGX," Proceedings 2018 Network and Distributed System Security Symposium, Jan. 2018.

[9] H. Oh, A. Ahmad, S. Park, B. Lee, and Y. Paek, "TRUSTORE: Side-channel resistant storage for SGX using Intel hybrid CPU-FPGA," Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 1903-1918, ov. 2020.

[10] A. Ahmad, B. Joe, Y. Xiao, Y. Zhang, I. Shin, and B. Lee, "Obfuscuro: A commodity obfuscation engine on Intel SGX," Proceedings 2019 Network and Distributed System Security Symposium, Feb. 2019.

[11] A. Shafiee, R. Balasubramonian, M. Tiwari, and F. Li, "Secure DIMM: Moving ORAM Primitives Closer to Memory," in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 428 - 440, Feb. 2018.

[12] S. Aga and S. Narayanasamy, "InvisiMem: Smart memory defenses for memory bus side channel," in 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 94-106, Jun. 2017.

[13] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices," ACM SIGPLAN Notices, vol. 53, no. 2, pp. 316–331, Mar. 2018.

[14] L. Ke, U. Gupta, B. Y. Cho, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, X. Wang, B. Reagen, C.-J. Wu, M. Hempstead, and X. Zhang, "RecNMP: Accelerating personalized recommendation with near-memory processing," 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), pp. 790-803, Sep. 2020.

[15] J. H. Kim, S.-haeng Kang, S. Lee, H. Kim, W. Song, Y. Ro, S. Lee, D. Wang, H. Shin, B. Phuah, J. Choi, J. So, Y. G. Cho, J. H. Song, J. Choi, J. Cho, K. Sohn, Y. Sohn, K. Park, and N. S. Kim, "Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond," 2021 IEEE Hot Chips 33 Symposium (HCS), pp. 1-26, Aug. 2021.

[16] J. Nider et al., 'A Case Study of Processing-in-Memory in off-the-Shelf Systems', in 2021 USENIX Annual Technical Conference (USENIX ATC 21), pp. 117–130. Jul. 2021.

[17] J. McMahan, W. Cui, L. Xia, J. Heckey, F. T. Chong, and T. Sherwood, "Challenging on-chip SRAM security with boot-state statistics," 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp.101-105, May. 2017.

〈저자소개〉

신 수 환 (Suhwan Shin) 학생회원
2022년 2월: 성균관대학교 소프트웨어학과 졸업
2022년 3월~현재: 성균관대학교 소프트웨어학과 석사과정
〈관심분야〉 시스템보안, 하드웨어 보안, 클라우드 AI보안

이 호 준 (Hojoon Lee) 정회원
2010년 12월: The University of Texas at Austin 졸업
2013년 8월: KAIST 석사
2018년 2월: KAIST 박사
2019년 11월~현재: 성균관대학교 소프트웨어대학 교수
〈관심분야〉 정보보호, 프로그램 분석, 소프트웨어보안, 시스템보안, TEE, 클라우드 AI보안