Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.28, NO.1, Feb. 2018 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2018.28.1.61

멀티코어 환경에서 효율적인 엔트로피 원의 설계 기법*

김 성 겸,[1†] 이 승 준,[1] 강 형 철,[1] 홍 득 조,[2‡] 성 재 철,[3] 홍 석 희[1]
[1]고려대학교, [2]전북대학교, [3]서울시립대학교

An Approach to Constructing an Efficient Entropy Source on Multicore Processor*

SeongGyeom Kim,[1†] SeungJoon Lee,[1] HyungChul Kang[1]
Deukjo Hong,[2‡] Jaechul Sung,[3] Seokhie Hong[1]
[1]Korea University, [2]Chonbuk National University, [3]University of Seoul

요 약

다양한 장비의 인터넷 연결을 지향하고 있는 사물인터넷시대에서 암호기술의 사용을 위해 암호학적으로 안전한 난수생성은 중요 요구사항이다. 특히, 생성된 난수의 안전성과 연관된 엔트로피 원은 예측하기 어려운 잡음원을 위해 부가적인 하드웨어 로직을 사용하기도 한다. 비록 성능 측면에서 좋은 결과를 나타낼 수 있으나, 부가적인 리소스의 사용에 기인한 추가적인 전력 소비 및 면적문제 때문에 기존 자원을 최대한 활용하는 엔트로피 수집방법이 요구된다. 본 논문에서 제시하는 엔트로피 원은 멀티쓰레드 프로그램을 지원하는 환경에서 부가적인 장치 없이 공통적으로 사용 가능하므로 암호기술 구현에 있어 경량화의 어려움을 완화시킬 수 있다. 또한, 제안하는 엔트로피 원이 NIST SP 800-90B에서 제시한 난수발생기를 위한 엔트로피 입력원 테스트에서 높은 보안강도를 갖는 것으로 평가 되었다.

ABSTRACT

In the Internet of Things, in which plenty of devices have connection to each other, cryptographically secure Random Number Generators (RNGs) are essential. Particularly, entropy source, which is the only one non-deterministic part in generating random numbers, has to equip with an unpredictable noise source(or more) for the required security strength. This might cause an requirement of additional hardware extracting noise source. Although additional hardware resources has better performance, it is needed to make the best use of existing resources in order to avoid extra costs, such as area, power consumption. In this paper, we suggest an entropy source which uses a multi-threaded program without any additional hardware. As a result, it reduces the difficulty when implementing on lightweight, low-power devices. Additionally, according to NIST's entropy estimation test suite, the suggested entropy source is tested to be secure enough for source of entropy input.

Keywords: cryptography, entropy, entropy source, SEI(Source of Entropy Input), DRBG, NRBG, nonce, IoT(Internet of Things), data race condition

Received10. 27. 2017), Modified(12. 05. 2017), Accepted(12. 08. 2017)

* 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 대학ICT연구센터육성 지원사업의 연구결과로 수행되었음(IITP-2018-2015-0-00385)

† 주저자, jeffgyeom@korea.ac.kr
‡ 교신저자, deukjo.hong@jbnu.ac.kr(Corresponding author)

I. 서 론

20세기 후반까지 경제를 이끌었던 제 3차 산업혁명 이후에 초연결, 초지능을 특징으로 하는 제 4차 산업혁명이 도래 하였다. 특히, 현재는 컴퓨터만의 연결을 뛰어넘어 다양한 기기들이 인터넷에 연결되어 서로 통신하고 있는 사물인터넷(IoT, Internet of Things)시대이다. 사물간의 연결은 편리함을 제공하는 한편 공격자에게 노출되는 정보가 늘어나는 보안 이슈를 증폭시켰다. 정보의 유출 위협이 증가된 상황에서, 대부분의 정보보호시스템은 공격자가 민감 정보를 획득하는 것을 차단하기 위해 높은 보안 강도를 가진 난수를 사용하여 유출 되는 정보에 충분한 불확정성과 변동성을 가하고 있다.

난수는 암호기술에서 기본적으로 사용되는 키(key), 초기벡터(IV, Initial Vector), 넌스(nonce), 시드(seed), 챌린지(challenge) 등과 같은 매개변수(parameter)를 생성할 수 있는 있는 가장 안전한 방법이다[24]. 특히, 시드[3], 초기화벡터, 생성자의 지수 값[4], 마스킹(masking)[5] 등과 같이 작동시점마다 새로 생성된 난수를 사용해야만 하는 경우에, 난수발생기는 암호알고리즘의 보안 강도뿐 만 아니라 난수의 생성속도가 암호알고리즘의 성능까지 영향을 미칠 수 있다.

난수발생기의 보안 강도 및 생성속도는 사용하는 엔트로피 원(entropy source)의 성질 및 사용 시점에 따라서 다르다[1][7][25]. 엔트로피 원을 난수 생성 시점마다 사용해야 하는 비결정론적 난수발생기(NRBG, Non-deterministic Random Bit Generator)는 비효율적이기 때문에 잘 사용하지 않고, 결정론적 난수발생기(DRBG, Deterministic Random Bit Generator)를 주로 사용한다. 결정론적 난수발생기는 비결정론적 난수발생기와 다르게 난수 생성 시점마다 엔트로피 원을 사용하는 것이 필수적이지는 않지만 작동 초기화(instantiation) 및 리시딩(reseeding)과정을 위한 엔트로피 입력원(SEI, Source of Entropy Input)에 필요하다[1].

엔트로피 입력원을 생성할 수 있는 엔트로피 원은 난수발생기의 보안강도를 결정짓는 중요한 요소이다. 프로세스 번호, 상위 프로세스 번호, 시간 값을 통해 보안강도가 낮은 엔트로피 원을 사용한 Netscape V2.0 브라우저는 암호알고리즘의 취약점이 아닌 SSL(Secure Socket Layer)에서 사용한 난수발생기 때문에 공격이 성공되었다[26]. 때문에, 안전한 엔트로피 원의 설계를 위해 하드웨어 구성요소의 사용을 권고하고 있으나[27][28], 방사성 붕괴[29], 열 잡음[30], 양자역학[31], jitter 잡음[32] 등을 이용한 하드웨어 기반의 엔트로피 원은 소형장비에 가용성을 떨어트려, 사용에 어려움이 있다.

본 논문에서는 멀티쓰레드 프로그램에서 발생할 수 있는 데이터 경합상태(data race condition)을 이용한 엔트로피 원을 제안한다. 싱글코어에서 발생하는 데이터 경합상태를 이용한 소프트웨어 기반의 엔트로피 원이 제안 되었고[33][34], 멀티코어에서 발생한 예측하기 어려운 결과 값을 이용한 엔트로피 원이 고안[20]되었으나, 두 가지 방법 모두 생성하는 쓰레드의 개수가 많아 생성 속도를 보장이 어렵고, 연산을 통해 얻은 값이 높은 엔트로피를 갖는 것을 실험적으로만 보였다. 본 논문에서는 두 개의 쓰레드만 사용하는 간단한 기법을 통해 높은 엔트로피를 갖는 엔트로피 원을 제안하고, 제안된 방법을 수학적으로 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 NIST SP 800-90B에서 제시한 엔트로피 원 구현 모델 및 CPU구조에 따른 데이터 경합에 대해서 설명하고 이후 본 논문의 핵심적 내용인 멀티쓰레드 프로그램의 구현에서 발생할 수 있는 데이터 경합상태를 이용한 엔트로피원에 대하여 기술한다. 3장에서는 본 논문에서 제안하는 데이터 경합상태를 이용한 엔트로피 원을 제시한다. 마지막으로 다양한 장비에서 구현된 엔트로피 원의 평가 실험 결과를 4장에서 기술한 후 5장에서 논문의 결론을 맺는다.

II. 관련 연구

2.1 엔트로피

본 논문에서 사용하는 *엔트로피*의 정의는 Shannon 엔트로피 정의를 따른 것으로, 관측되지 않은(혹은 미래에 관측될) m bits의 크기를 가진 정보가 될 수 있는 값의 분포를 이산확률변수(discrete random variable) X로 표현 했을 때, 각 가능한 정보가 발생할 수 있는 확률의 분포 정도를 지칭 하는 것이다.

$$Entropy = \sum_{x \sim X} -p(x)\log_2(p(x))\, bits \qquad (1)$$

$$Min\,Entropy = -\log_2(\max_{(x \sim X)} p(x))\,bits \quad (2)$$

엔트로피의 수치화된 값을 얻어내기 위한 식은 (1)처럼 나타낼 수 있으며, $p(x)$는 정보의 값이 x가 될 수 있는 확률을 뜻한다. $m$bits 크기(혹은 2^m개의 가지 수)를 가진 정보의 엔트로피는 최대 $\log_2(2^m) = m$bits 가 될 수 있으나, $(1-\varepsilon)m$bits $(0 \leq \varepsilon \leq 2^{-64})$의 엔트로피 크기를 *최대 엔트로피(full entropy)*라 지칭 한다. 정보의 *최소 엔트로피(min entropy)*는 (2)처럼 나타낼 수 있으며 나올 수 있는 정보의 값 중 가장 높은 확률을 가지는 정보 값이 그 값을 결정한다.

2.2 엔트로피 원 모델 및 평가도구

*엔트로피 원*은 예측불가능한 잡음원으로 부터 수치화된 데이터를 출력하는 일련의 과정이다. *실시간 엔트로피 원(live entropy source)*은 실시간으로 특정 엔트로피 크기의 데이터를 출력할 수 있는 엔트로피 원을 지칭하며 비결정론적 난수발생기 구현에 있어서 핵심이 되는 요소이다. *엔트로피 입력원*은 결정론적 난수발생기의 작동 초기화 및 리시딩 시점에서 필요한 엔트로피 입력으로 사용할 수 있는 것으로 엔트로피 원을 통해 얻을 수 있다[7]. 결국 비결정론적, 결정론적 난수발생기는 엔트로피 원의 출력 값이 있어야만 구현이 가능하며 엔트로피 원이 갖는 엔트로피 크기는 난수발생기의 보안강도를 결정하기[1] 때문에 일정수준의 엔트로피 크기를 보장할 수 있는 엔트로피 원을 사용해야만 안전한 난수발생기 구현이 가능하다. 따라서 NIST가 권고하는 엔트로피 원 모델에는 낮은 엔트로피를 보정해주는 컨디셔닝 과정이 포함되어 있다.

Fig. 1은 엔트로피 원의 세부 과정을 도식화 한 것으로 시간, 온도, 메모리 사용량, 프로세스 정보 등과 같은 다양한 잡음원을 수치화 하는 과정과 컨디셔닝 과정으로 나뉜다. 수치화된 데이터 자체의 엔트로피는 근본적으로 잡음원에서 오기 때문에 높은 엔트로피를 기대할 수 있는 가용성이 높은 잡음원을 이용하는 것이 좋은 엔트로피 원을 구성할 수 있는 밑바탕이 된다.

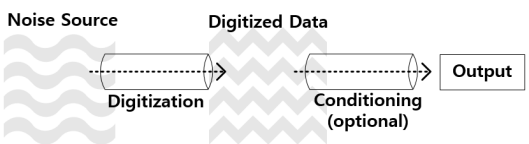

Fig. 1. Entropy source model

엔트로피 원의 평가는 NIST SP 800-90B[2]에서 제시된 엔트로피 크기 추정 방법을 통해 가능하다. 엔트로피 원의 출력이 IID(Independent Identical Distribution)를 만족하는지 아닌지로 추정 방법이 다르며, 추정 이전에 IID 테스트를 먼저 실시한다. 보통의 물리적 현상을 통한 잡음원은 시간에 따라 연속된 수치를 갖기 때문에 IID를 만족하기 어렵다. 이러한 경우, Non-IID 엔트로피 원으로 분류되어 10가지 추정방법에서 얻어낸 엔트로피 크기 중 가장 작은 값으로 엔트로피 원이 평가 받게 된다.

2.3 멀티코어 프로세서와 데이터 경합상태

본 논문에서 지칭하는 *멀티코어 프로세서*는 Fig. 2에서 도식화 한 것과 같이 물리적으로 독립된 둘 이상의 코어(core)가 하나의 칩을 이루고 있는 프로세서를 의미한다. 2000년대에 CPU의 가속화의 비효율성 문제로 멀티코어 프로세서의 디자인이 대두 되어 현재 대부분의 CPU가 채택하고 있는 디자인이다[8]. 또한 멀티코어 프로세서기반 프로그래밍에서 쓰레드 수준 병렬성(thread-level parallelism, TLP)이란 하나의 멀티코어 프로세서에서 둘 이상의 쓰레드(thread)를 실행 시킬 수 있는 병렬 수준을 의미하며 인스트럭션 수준 병렬성(instruction-level parallelism)과 다르게 코어 개수만큼의 인스트럭션을 동시에 처리 가능한 병렬수준을 뜻한다.

쓰레드 수준 병렬성을 통해 구현된 프로세스

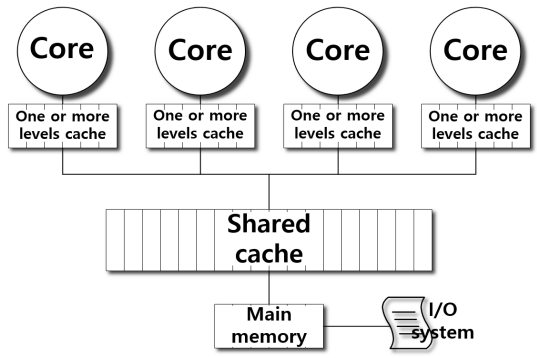

Fig. 2. Basic structure of a centralized shared-memory multiprocessor based on a multicore chip

(process)는 물리적으로 독립된 다수의 코어를 동시에 사용하면서 캐쉬, I/O 등의 자원을 공유한다. 공유 캐쉬(i7의 경우, L3)는 설계된 캐쉬 정책(cache coherence protocol)을 통해 각 코어의 캐쉬(i7의 경우, L1과 L2)와 캐쉬 일관성(coherence)을 유지하고 있다[8]. 특히, 동일한 캐쉬 메모리 공간에 다수의 동시접근이 생기는 상태를 데이터 경합상태 라고하며 캐쉬 일관성을 위해 상호배제(mutual exclusion)방식을 통해 다수의 접근이 모두 실행될 수 있도록 하고 있다. 접근요청이 무시가 될 수 없도록 상호배제방식을 취할 수 있으나 데이터 경합상태가 발생하면 프로그래머가 의도하지 않은 값을 메모리에 저장하는 경우가 발생할 수 있다.

예를 들어 공유 변수 X에 3과 2를 더하는 두 개의 쓰레드로 구성된 프로세스가 실행되어 Table 1과 같이 다른 코어에 각각 쓰레드가 배치될 경우 의도했던 값 5가 더해지는 것이 아닌 3만 더해진 값을 얻을 수도 있다. Fig. 3은 두 개의 쓰레드의 연산한 결과 값이 원래 의도된 값(30,000)이 아닌 값을 출력하는 것을 보여주고 있으며 그 결과 값은 1,000,000번의 실행마다 달랐다. 이와 같이 의도하지 않은 연산 결과를 초래하는 데이터 경합은 병행성 버그(concurrency bug)로 취급되어 쓰레드 수준의 병렬 프로그램에서 발생가능지점을 탐지하기 위한 여러 기법들이 연구 되고 있다[11-13].

Table 1. A scenario of data race condition which causes an unpredictable output

Core #1		Core #2		Shared cache
Instructions	Cache X =	Instructions	Cache X =	X =
mov ebx, X	read miss			100
…	delay	mov ebx, X	read miss	100
…	100	…	delay	
add ebx, 2	100	…	100	
mov X, ebx	102	add ebx, 3	100	102
		mov X, ebx	103	103

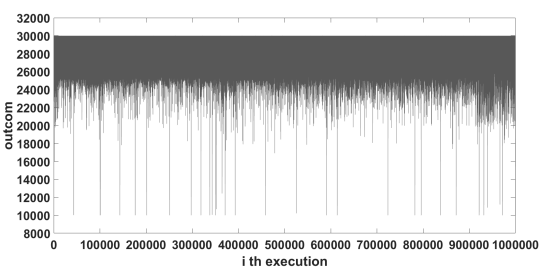

Fig. 3. Unpredictable outputs from 1,000,000 executions of 10,000 additions of 1,2 to shared memory using two threads in a process

2.4 공유 메모리 다중 처리 프로그래밍 API

엔트로피 원을 구현하기 위해 사용한 공유 메모리 다중 처리 프로그래밍 API는 Pthread[14], Windows API, OpenMP[15] 이다. 각기 다른 함수 형태를 지니고 있지만 근본적으로 하나의 프로세스에서 여러 개의 쓰레드를 생성하고 동기화하기 위한 방법을 제공하는 함수를 가지고 있다. Pthread와 Windows API의 경우 각각 Linux, Windows 운영체제에 맞는 쓰레드 수준의 병렬 프로그램을 작성할 수 있으며, 어셈블리 코드를 가진 쓰레드를 생성시키는 것이 가능하다. 반면 OpenMP는 모든 계열의 운영체제에서 사용이 가능하지만 병렬 구간 내에서 어셈블리 코드를 사용할 수 없다. 각각의 API는 쓰레드의 동기화를 위해 Fig. 4에서 도식한 개념의 *베리어(barrier)*를 사용하고 있으며, 하나의 베리어 안에 속한 쓰레드들은 분배된 모든 작업이 끝날 때까지 서로를 대기하게 된다.

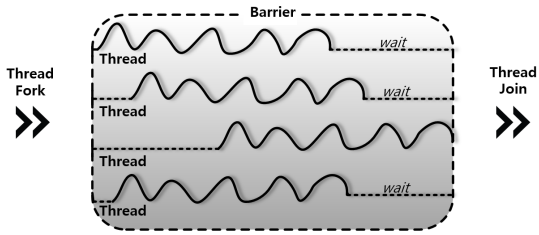

Fig. 4. The use of barrier to synchronize threads.

III. 데이터 경합상태 유도 엔트로피 원

본 장에서는 데이터 경합을 잡음원으로 하여 엔트로피 원을 구하는 방법을 제시한다.

3.1 사용 잡음원

잡음원은 [2]에서 정의한 바와 같이 값을 얻기 전에 어떤 값이 될지 예상할 수 없어야 한다. 데이터 경합이 일어날 수 있는 시점을 포함한 연산은 예측하기 어려운 결과 값을 출력하기 때문에 잡음원으로 사

용할 수 있는 비예측성(unpredictability)을 만족할 수 있다. 이와 같은 데이터 경합을 잡음원으로 하는 엔트로피원은 이전에도 제시 되었는데, 높은 엔트로피를 보장하기 위해 [20]과 같이 높은 연산 복잡도와 많은 개수의 쓰레드 사용을 통해 얻은 결과 값을 출력으로 하는 엔트로피 원을 사용 할 수 있으나, 이러한 방법은 많은 리소스를 요구한다. 하지만, 리소스 요구량을 줄이기 위해 단순한 연산을 하는 적은 개수의 쓰레드를 통해 얻은 결과 값을 그대로 사용하면 높은 엔트로피를 보장할 수 없다. 단순한 더하기 연산을 한 결과 값의 통계치를 보여주는 Fig. 5에서 추정할 수 있는 결과 값의 최소 엔트로피 값은 $-\log_2(0.000334) \approx 11.5479 bits$로서 최대 엔트로피 $\log_2(30001) = 14.8727 bits$에 비해 작다. [22]에서는 비교적 단순한 연산을 하면서도 높은 엔트로피 보장을 위해 결과 값을 이용하여 1bit의 값을 추출하는 것을 제안하였으나, 데이터 경합 발생 확률을 높이기 위해 다수의 쓰레드 사용이 필수적이다.

3.2절과 3.3절에서는 단순한 연산을 하면서도 높은 엔트로피를 얻을 수 있는 방법을 제시하고 수학적으로 증명한다.

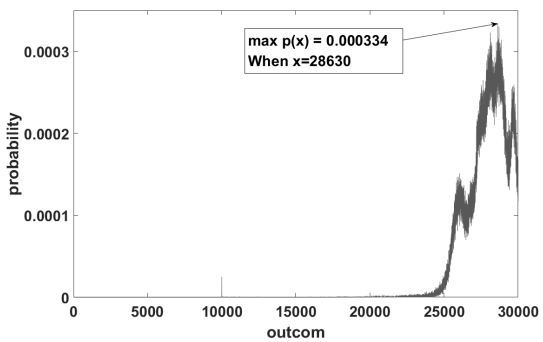

Fig. 5. Probability distribution of outcomes (except when outcome=30,000) from 1,000,000 executions of 10,000 additions of 1,2 to shared memory using two threads in a quad-core process.

3.2 데이터 경합상태 유도 엔트로피 원

Fig. 6은 데이터 경합 이후에 **Shared_mem**에 저장된 값에 따라서 0 또는 1을 출력하는 알고리즘을 보여준다. **Shared_mem**이 3의 배수가 될 확률을 제외하고, **Shared_mem**을 3으로 나눈 나머지가 1 또는 2가 될 확률이 동일한 것을 이용한 방법으로 **Shared_mem**에 3의 배수가 저장되어 있다면 3의 배수가 나오지 않을 때 까지 데이터 경합을 재시도 한다.

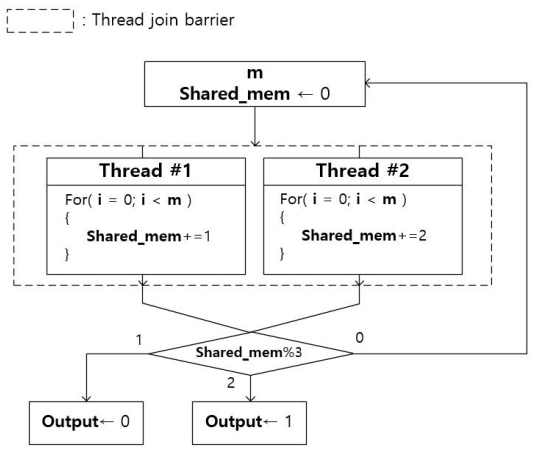

Fig. 6. Entropy source using data race condition as the noise source. In a thread join barrier, threads should be forked at the same time and wait each other until they finish their job.

Shared_mem이 3의 배수가 될 확률을 제외하고 3으로 나눈 나머지가 1 또는 2가 될 확률이 동일하다는 수학적 분석을 위해 정의 1과 같이 확률변수를 정의한다.

정리 3에 의해서 Fig. 6의 알고리즘의 출력이 0 또는 1일 확률이 동일함을 보일 수 있으며, 매 출력은 서로 독립하다고 볼 수 있으므로 최대 엔트로피를 갖는 엔트로피 원을 구성 할 수 있다.

정의 1.
Fig. 5의 두 쓰레드(**Thread #1, Thread #2**)가 각각 **m**번의 연산을 실행 하는 동안 발생한 데이터 경합에서 다음의 확률 변수를 정의 할 수 있다.
X : **m**번의 더하기 연산 중 **Shared_mem**에 1을 더한 횟수
Y : **m**번의 더하기 연산 중 **Shared_mem**에 2를 더한 횟수

가정 2.
정의 1에서 정의한 확률변수 X, Y는 다음을 만족한다.
$P(X=x, Y=y) = P(X=y, Y=x) \quad \forall x,y$

정리 3.
Fig. 5의 알고리즘은 0 또는 1을 동일한 확률로 출력한다.
〔증명〕 후략

3.3 3.2절의 알고리즘에 대한 편향 테스트 결과 및 분석

편향 테스트(Bias Test)
편향 테스트는 정리 3에서 도출한 바와 같이 0 또는 1이 동일한 확률로 나오는지 확인하기 위해 10,000 비트를 생성하여 0과 1의 개수를 측정한다.

3.2절의 알고리즘을 통해 얻은 출력 값의 평가를 위해 NIST SP 800-90B에서 제공되는 소프트웨어를 사용할 수 있겠으나, 평가 이전에 간단한 **편향 테스트**를 통해 0또는 1이 나오는 확률을 확인 할 수 있다.

정리 3과 Fig. 5에서 확인 할 수 있듯이 생성된 쓰레드가 각각 1 또는 2를 더하는 횟수 **m** 은 0또는 1이 나오는 확률과 관련이 없다. 따라서 Table 4에서 환경별로 초당 출력비트의 크기가 가장 클 때의 **m** 값을 사용하여 **편향 테스트**를 시행 하였다. 컴파일러의 최적화 옵션을 제거하여 프로그램이 코드 그대로 실행할 수 있도록 하였다.

편향 테스트를 1,000번 반복해서 얻은 결과 값(Fig. 7)에서 편향된 값을 가지는 몇몇 환경을 확인할 수 있었다. 이는 가정 2를 만족하지 못하는 경우에 해당된다. Fig. 8와 같이 병렬화 구간 전의 주 쓰

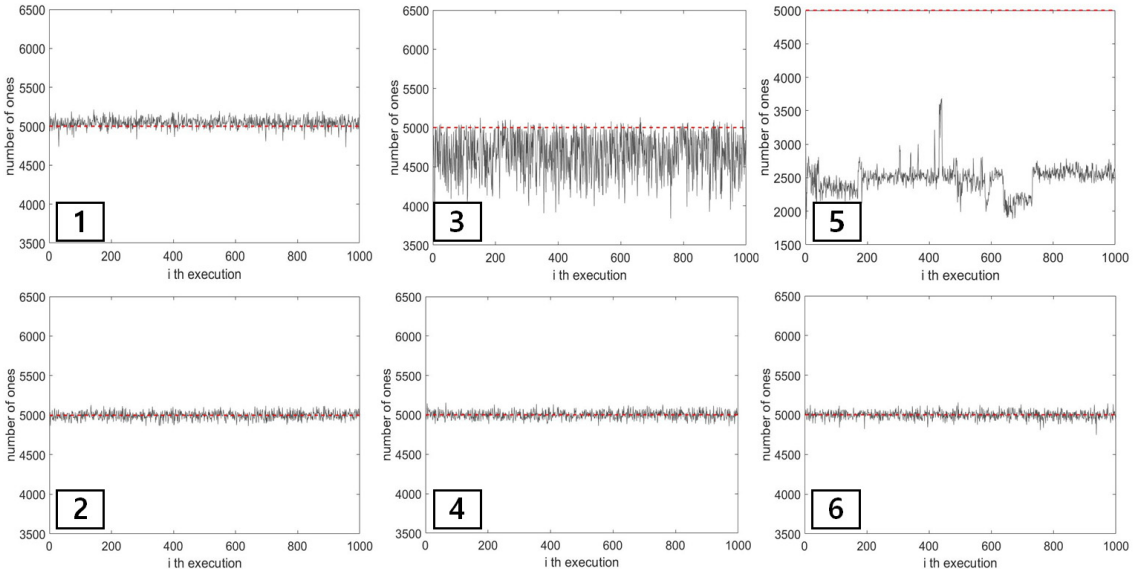

Fig. 7. Bias graphs in the four different environments. The number of ones means the hamming weight in a 10,000bits string on each execution. The red lines locate at the half of 10,000(ie, 5,000).

Table 2. The information of a parameter , environment used and speed using m respectively in the bias test. The m means the number of additions 1 or 2 in Fig. 6.

	Device	Processor	OS	Multi process API	**m =**	Speed (Bit/s)
1	Personal Computer	Intel i7-4770	Windows 10 Home	OpenMP	140	659,889
2	Personal Computer	Intel i7-4770	Windows 10 Home	Windows API	18,000	3,566
3	Surface Pro4	Intel m3-6y30	Windows 10 Pro	OpenMP	170	315,303
4	Surface Pro4	Intel m3-6y30	Windows 10 Pro	Windows API	10,500	1424
5	Raspberry pi 3	ARMv7 Processor rev 4	Raspbian minimal〔16〕	OpenMP	100	207,517
6	Raspberry pi 3	ARMv7 Processor rev 4	Raspbian minimal	PThread	1,000	5989

레드가 병렬화 구간에서 부 쓰레드를 생성하는 방식으로 코드 상에서 첫 번째 쓰레드가 두 번째 쓰레드와 동시에 시작 하지 못하고 먼저 시작하는 경우에 가정 2를 만족 시킬 수 없다. 하지만, Fig 7에서 확인할 수 있듯이 1이 나올 확률이 어느 정도 유지된다는 것을 유추할 수 있다.

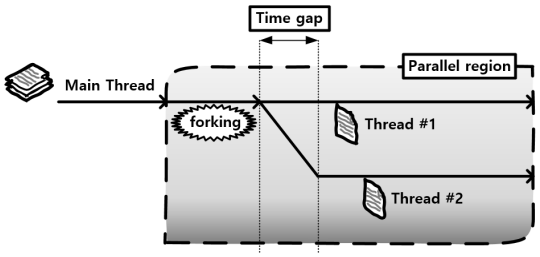

Fig. 8. The time gap causes 가정 2 to fail.

3.4 편향 보정 방법이 적용된 데이터 경합상태 유도 엔트로피 원

Thread #1,#2의 시작 시간의 차이로 인해 편향된 출력을 보정하기 위해 두 번의 경합 유도를 거친다 (Fig. 9). 첫 번째 경합 유도 과정(첫 번째 Thread join barrier)에서 **Thread #1,#2**가 각각 1,2를 공유된 메모리에 더하기 연산을 **m**번 반복하고, 두 번째 경합 유도 과정(두 번째 Thread join barrier)에서 첫 번째 유도 과정과는 반대로 **Thread #1,#2**가 각각 2,1을 더하기 연산을 **m**번 반복한다. 두 번의 경합 유도 과정에서 더하는 값이 서로 크로스(cross)되어 편향된 확률을 보정해 줄 수 있다. Fig. 10은 Fig. 7의 편향 그래프와 비교한 것으로 이전에 편향된 결과를 가졌던 1,3,5 환경에서 향상된 결과를 보여주고 있다.

이러한 방법은 PUFs(Physical Unclonable Functions)[23]기반 키 출력의 편향문제를 해결하기 위해서 [18]에서 제시된 편향 보정 기법과 유사한 것으로 경합 유도 과정을 여러 번 사용하면 편향성의 보정효과가 커진다. Table 3에서 확인 할 수 있듯이, 다수의 경합유도 과정은 연산량의 증가로 인해 비트 생성속도에 영향을 미치지만 2배의 경합 유도 과정을 거친다고 해서 생성속도가 절반으로 줄지는 않는다.

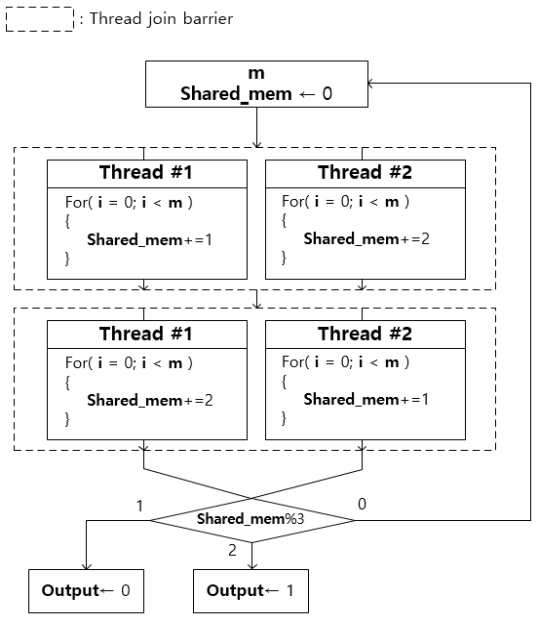

Fig. 9. Improved entropy source causing data race as the noise source. In a thread join barrier, threads are forked at the same time and wait each other until they finish their job.

IV. 데이터 경합상태 유도 엔트로피 원의 평가

Table 4는 개선한 Fig.9의 방법을 통해 만들어진 엔트로피 원에 대해서 Table 3의 여섯 가지 환경에서 구성한 엔트로피 원의 평가를 NIST SP 800-90B에서 제공되는 소프트웨어[17]를 통해 측정한 결과이다. 이 소프트웨어는 엔트로피 원으로부터 얻은 8비트 이하의 샘플들에 대해서 그 엔트로피를 평가하는 것으로 본 논문에서는 8비트의 샘플 1,000,000개를 각각의 환경에서 생성하여 평가 하였다. IID Test를 통해 앞서 언급한 바와 같이 IID라고 판정된 잡음원은 상대적으로 높은 Min Entropy 값으로 측정 될 수 있는데, 6개의 실험 환경 중 OpenMP API를 사용하고 있는 환경 5에서만 IID Test를 통과하지 못하여 가장 낮은 Min Entropy 측정값을 가졌다. 그러나 Fig. 9에서의 Barrier 개수를 더 늘려서 편향된 확률 값을 보정한 뒤, 실험하였을 때 IID Test의 통과와 동시에 다른 환경과 같이 높은 Min Entropy(8개의 Barriers, 7.89633 bits)가 측정 되었다.

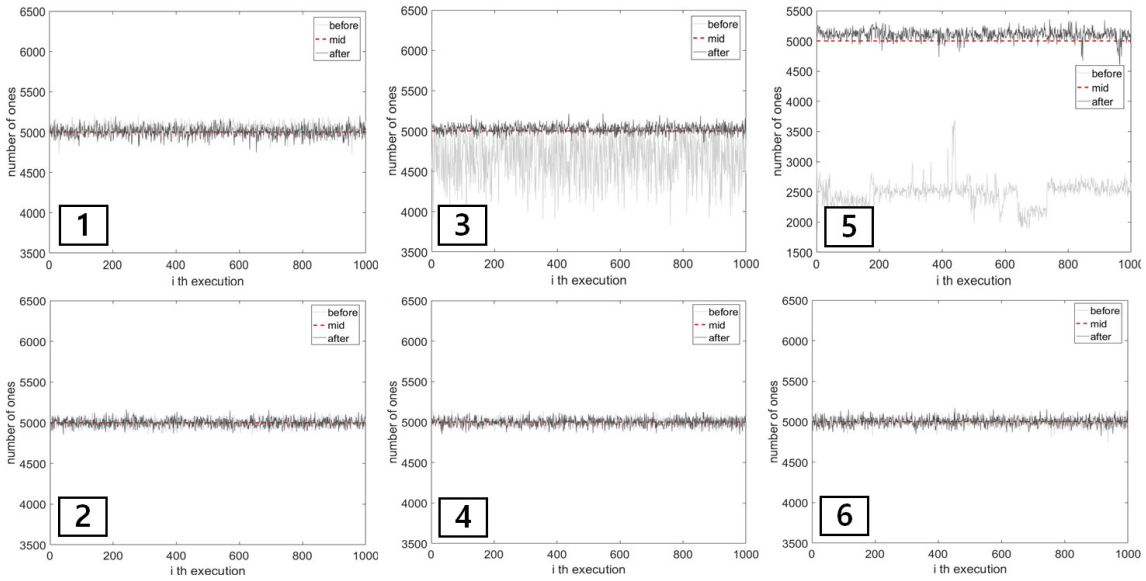

Fig. 10. Bias graphs in the four different environments using the improved entropy source. The number of ones means the hamming weight in a 10,000bits string on each execution. The red lines locate at the half of 10,000(ie, 5,000).

V. 결 론

본 논문에서는 멀티 쓰레드 프로그램에서 버그로 알려진 데이터 경쟁 상태를 이용한 엔트로피 입력원을 제안하였다. 또한, NIST SP 800-90B의 엔트로피 측정 평가를 이용하여 제안한 엔트로피 원의 출력 데이터가 높은 엔트로피를 갖는 것을 확인하였다. 일반 개인용 컴퓨터 뿐만 아니라 실제 임베디드 프로세서 상에서 실험을 수행함으로써, 잡음원 수집여건이 제한된 환경에서도 제안한 편향 보정 방법을 통해 높은 보안강도의 엔트로피 원을 얻을 수 있다는 것을 확인 할 수 있었다. 향후 연구에서는 본 논문에서 제안한 수집방법을 싱글 코어 프로세서기반의 실험을 통해 Low-cost의 마이크로컨트롤러에서의 유용성을 검증할 예정이다.

정리 3의 증명.
확률변수
Z : 경합 이후 **Shared_mem**에 저장된 값
이라고 하면 다음을 만족한다.
$Z = X + 2Y \equiv X - Y \mod 3$
$P(Z \equiv 2 \bmod 3)$
$= P(X \equiv 0 \text{ and } Y \equiv 1 \bmod 3)$
$\quad + P(X \equiv 1 \text{ and } Y \equiv 2 \bmod 3)$
$\quad + P(X \equiv 2 \text{ and } Y \equiv 0 \bmod 3)$
가정 2에 의해서
$= P(X \equiv 1 \text{ and } Y \equiv 0 \bmod 3)$
$\quad + P(X \equiv 2 \text{ and } Y \equiv 1 \bmod 3)$
$\quad + P(X \equiv 0 \text{ and } Y \equiv 2 \bmod 3)$
$= P(X - Y \equiv 1 \bmod 3)$
$= P(Z \equiv 1 \bmod 3)$
$\therefore P(Z \equiv 1 \bmod 3) = P(Z \equiv 2 \bmod 3)$ 이고
$Z \equiv 0 \bmod 3$ 일 때는 결과를 출력 하지 않으므로 0 또는 1을 동일한 확률로 출력한다. □

References

[1] Barker, E., and et al, "Recommendation for random number generation using deterministic random bit generators (revised)," NIST SP 800-90A, US Department of Commerce, NIST, 2007.

[2] Barker, E., and et al, "NIST DRAFT Special Publication 800-90b recommendation for the entropy sources used for random bit generation," NIST SP 800-90B, US Department of Commerce, NIST, 2012.

[3] Jonsson, Jakob, and et al, RSAES-OAEP

Encryption. "PKCS# 1: RSA Cryptography Specifications Version 2.2," PKCS# 1, RSA Laboratories, 2016.
〔4〕 Gallagher and Patrick, "Digital signature standard (DSS)," FIPS PUB 186-3, June. 2013.
〔5〕 Messerges and et al, "Power analysis attacks of modular exponentiation in smartcards," CHES, pp. 144-157, Aug. 1999.
〔6〕 Hojoong Park, Ju-Sung Kang, and Yongjin Yeom, "Probabilistic Analysis of AIS.31 Statistical Tests for TRNGs and Their Applications to Security Evaluations," Journal of The Korea Institute of Information Security & Cryptology, 26(1), pp. 49-67, Feb. 2016.
〔7〕 Barker, E., and et al, "Recommendation for Random Bit Generator (RBG) Constructions," NIST SP 800-90C, US Department of Commerce, NIST, 2012.
〔8〕 Hennessy and et al, Computer architecture: a quantitative approach. Elsevier, 6th Ed., Patterson, Nov. 2011.
〔9〕 Gutterman and et al, "Analysis of the linux random number generator," IEEE Symposium on Security and Privacy, pp. 370-385, May. 2006.
〔10〕 Dorrendorf and et al, "Cryptanalysis of the random number generator of the windows operating system," ACM Transactions on Information and System Security, vol.13, no.1, Oct 2009.
〔11〕 Shao and et al, "Data Race Detection by Understanding Synchronization Relationships of Thread Segments," IEEE PDP, pp. 229-232, March. 2017.
〔12〕 Yang and et al, "RaceTracker: Effective and efficient detection of data races," IEEE SNPD, pp. 223-300, June. 2016.
〔13〕 Ibing and Andreas, "Efficient data-race detection with dynamic symbolic execution," IEEE FedCSIS, pp. 1719-1726, Sep. 2016
〔14〕 "Portable Operating System Interface (POSIX) Base Specifications, Issue 7," ISO/IEC/IEEE 9945:2009, Sep. 2009.
〔15〕 Dagum and et al, "OpenMP: an industry standard API for shared-memory programming," IEEE computational science and engineering, vol. 5, no. 1, pp. 46-55, Mar. 1998
〔16〕 https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
〔17〕 Hall and et al, "User's Guide to Running the Draft NIST SP 800-90B Section 9 Entropy Estimation Tests," Mar. 2015.
〔18〕 Maes and et al, "Secure key generation from biased PUFs," CHES, pp. 517-534, Sep. 2015
〔19〕 Rukhin, Andrew, and et al, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST SP 800-22, Statistical Engineering Division and Computer Security Division, NIST, April. 2010.
〔20〕 "Method for entropy harvesting for random number generators in multi-core computing environments," patent application number 10-2014-0017136, Oct. 2015
〔21〕 Lim and et al, "Extracting secret keys from integrated circuits," IEEE Transactions on VLSI, vol.13, no.10, pp. 1200-1205, Dec. 2005
〔22〕 Voldman and Jean, "Method and system for generating actual random numbers within a multiprocessor system," U.S. Patent No. 6,487,571. 26, Nov. 2002.
〔23〕 Lin and et al, "Low-power sub-threshold design of secure physical unclonable functions," IEEE ISLPED, pp. 43-48, Aug. 2010.
〔24〕 Marton and et al, "Randomness in digital cryptography: A survey," Romanian Journal of Information Science and Technology, vol. 13, no. 3, pp. 219-240, 2010.
〔25〕 Suciu and et al, "Unpredictable random

number generator based on hardware performance counters," Digital Information Processing and Communications, 2011. pp. 123-137, 2011.
〔26〕 Goldberg and et al, "Randomness and the Netscape browser," Dr Dobb's Journal-Software Tools for the Professional Programmer, vol. 21, no. 1, pp. 66-71, 1996.
〔27〕 Eastlake 3rd, D., J. Schiller, and et al, "Randomness requirements for security," RFC 4086, 2005.
〔28〕 Ellison and Carl, "Cryptographic random numbers," Draft P1363 Appendix E, 2007.
〔29〕 Walker and John, "HotBits: Genuine random numbers, generated by radioactive decay," 2001.
〔30〕 Chindris and et al, "Bipolar junction effects for high entropy data harvesters," IEEE SYNASC, pp. 449-452, Sep. 2008.
〔31〕 Stefanov, Andre, and et al, "Optical quantum random number generator," Journal of Modern Optics, vol. 47, no. 4, pp. 595-598, Sep. 1999.
〔32〕 Simka, Martin, and et al, "Model of a true random number generator aimed at cryptographic applications," IEEE ISCAS, pp. 21-24, May. 2006.
〔33〕 Colesa and et al, "Software random number generation based on race conditions," IEEE SYNASC, pp. 439-444, Sep. 2008.
〔34〕 https://ark.intel.com/products/27447/Intel-Pentium-4-Processor-2_80-GHz-512K-Cache-533-MHz-FSB

〈저자소개〉

김 성 겸 (SeongGyeom Kim) 학생회원
2016년 8월: 한양대학교 수학과 졸업
2016년 9월~현재: 고려대학교 정보보호대학원 석사과정
〈관심분야〉 암호 알고리즘 설계 및 분석, 대칭키 암호, 난수발생기

이 승 준 (SeungJoon Lee) 학생회원
2013년 2월: 한림대학교 컴퓨터공학과 졸업
2013년 9월~현재: 고려대학교 정보보호대학원 석박사통합과정
〈관심분야〉 암호화 하드웨어 셀계, Secure Logic Cell, FPGA 보안, 임베디드 시스템

강 형 철 (Hyngchul Kang) 학생회원
2010년 2월: 고려대학교 산업시스템정보공학과 학사 졸업
2018년 2월: 고려대학교 정보보호대학원 석박사통합
2018년 2월: 삼성
〈관심분야〉 블록 암호화 해쉬 함수 설계 및 분석, 인증 암호화 모드 설계

홍 득 조 (Deukjo Hong) 종신회원
1999년 8월: 고려대학교 수학과 학사
2001년 8월: 고려대학교 수학과 석사
2006년 2월: 고려대학교 정보보호대학원 박사
2006년 3월~2007년 12월: 고려대학교 정보보호기술연구센터 연구교수
2007년 12월~2015년 8월: 국가보안기술연구소 선임연구원
2015년 9월~현재: 전북대학교 IT정보공학과 조교수
〈관심분야〉 암호 알고리즘 설계 및 분석

성 재 철 (Jaechul Sung) 종신회원
1997년 8월 : 고려대학교 수학과 학사
1999년 8월 : 고려대학교 수학과 석사
2002년 8월 : 고려대학교 수학과 박사
2002년 8월~2004년 1월 : 한국정보보호진흥원 선임연구원
2004년 2월~현재 : 서울시립대학교 수학과 전임강사, 조교수, 부교수, 교수
〈관심분야〉 암호 알고리즘 설계 및 분석

홍 석 희 (SeokHie Hong) 종신회원
1995년: 고려대학교 수학과 학사
1997년: 고려대학교 수학과 석사
2001년: 고려대학교 수학과 박사
1999년 8월~2004년 2월: ㈜시큐리티 테크놀로지 선임연구원
2003년 3월~2004년 2월: 고려대학교 정보보호기술연구센터 선임연구원
2004년 4월~2005년 2월: K.U. Leuven ESAT/SCD-COSIC 박사후 연구원
2005년 3월~2013년 8월: 고려대학교 정보보호대학원 부교수
2013년 9월~현재: 고려대학교 정보보호대학원 정교수
〈관심분야〉 대칭키 및 공개키 암호 알고리즘, 부채널 공격 및 대응기법, 디지털 포렌식