Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.31, NO.2, Apr. 2021 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2021.31.2.145

퓨시아 운영체제 및 지르콘 커널의 난수발생기에 대한 안전성 분석 연구*

김 예 원,[1†] 염 용 진,[2‡] 강 주 성[2]
[1]국민대학교 금융정보보안학과 (대학원생), [2]국민대학교 정보보안암호수학과 (교수)

Security Analysis of Random Number Generator in Fuchsia Operating System and Zircon Kernel*

Yewon Kim,[1†] Yongjin Yeom,[2‡] Ju-Sung Kang[2]
[1]Dept. of Financial Information Security, Kookmin University (Graduate student),
[2]Dept. of Information Security, Cryptology and Mathematics, Kookmin University (Professor)

요 약

퓨시아(Fuchsia)는 구글(Google)이 데스크탑에서부터 IoT 기기까지의 환경에서 사용될 수 있도록 공개적으로 개발하고 있는 운영체제이다. 이는 구글이 개발하고 있는 지르콘(Zircon)이라는 마이크로 커널(microkernel)을 기반으로 한다. 암호키, 보안 매개변수 등을 생성하는 데 사용되는 난수발생기의 취약성은 운영체제의 안전성에 영향을 미치므로, 퓨시아 및 지르콘에서 사용하는 난수발생기에 대한 안전성 분석이 필요하다. 본 논문에서는 퓨시아 및 지르콘의 난수발생기에 대한 구조와 사용하는 엔트로피 소스를 분석한다. 또한, 퓨시아 및 지르콘의 난수발생기의 유일한 입력이 될 가능성이 큰 지터 엔트로피 소스에 대한 엔트로피를 추정하고, 지터 엔트로피 소스에 대한 전수조사 방법과 전수조사량을 제시한다. 본 논문의 결과는 다양한 환경에서 동작하는 퓨시아 및 지르콘 난수발생기에 대한 공격 가능성을 보여주며, 지터 엔트로피 소스를 사용하는 여러 운영체제의 난수발생기에 대한 안전성을 분석하는 데 활용될 것으로 기대한다.

ABSTRACT

Fuchsia is an operating system that is being developed publicly by Google to be used in environments ranging from desktops to IoT devices. It is based on a microkernel called Zircon that Google is developing. Since the vulnerability of the random number generator(RNG) used to generate cryptographic keys and security parameters affects the security of the operating system, it is necessary to analyze the security of the RNG used in Fuchsia and Zircon. In this paper, we analyze the structure of the RNG used in Fuchsia and Zircon and the entropy source used by RNG. In addition, we estimate the entropy of the jitter entropy source, which is likely to be the only input of the RNG of Fuchsia and Zircon, and propose a method and a cost of exhaustive search for the jitter entropy source. The results of this paper show the possibility of attacking RNG of the Fuchsia and Zircon operating in various environments and are expected to be used to analyze the security of RNGs of various operating systems that use the jitter entropy source.

Keywords: Fuchsia operating system, Zircon kernel, Jitter entropy source, Random number generator

Received(01. 20. 2021), Accepted(02. 24. 2021)
* 본 연구는 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행하였습니다. (No.2014-6-00908, 난수발생기 및 임베디드 기기 안전성 연구)
† 주저자, fdt150@kookmin.ac.kr
‡ 교신저자, salt@kookmin.ac.kr(Corresponding author)

I. 서 론

퓨시아(Fuchsia)는 지르콘(Zircon)이라는 마이크로 커널(microkernel)을 기반으로 하는 운영체제이다. 퓨시아는 2016년 8월 깃허브(GitHub)에 코드가 게시된 것으로 공개되었고, 지난 4년 동안 구글(Google)은 깃 저장소에서 공개적으로 퓨시아와 지르콘을 개발해 왔다. 2019년 1월, 구글은 퓨시아에 대한 설명과 설치 방법 등을 제공하는 홈페이지[1]를 공개하였다. 또한, 2020년 8월에 구글은 외부 개발자도 퓨시아 개발 프로젝트에 참여할 수 있도록 개방하였다[2]. 이를 위해, 구글은 퓨시아 개발 프로젝트의 메일링리스트(mailing list), 거버넌스 모델(governance model), 이슈 트래커(issue tracker), 기술 로드맵(roadmap) 등을 공개하였고, 개발 목적인 보안성, 업데이트성, 포괄성과 실용성에 관한 핵심 아키텍처(architecture) 원칙도 공개하였다. 퓨시아는 구글에서 발표한 사용처가 없으나, IoT 기기부터 데스크톱에서 실행될 수 있도록 개발되고 있다.

난수(random number)는 운영체제와 커널에서 사용하는 암호키, 논스(nonce), 보안 매개변수 등을 구성하는 필수 요소이다. 난수를 생성하는 난수발생기(Random Number Generator, RNG)는 잡음원(noise source)을 입력으로 사용하고 결정론적 알고리즘으로 구성되어 있다. 그러므로, 잡음원이 가진 엔트로피(entropy)가 충분하지 않으면 출력인 난수가 예측 가능해져 인가되지 않은 사용자가 중요 데이터를 얻을 수 있다. 잡음원의 엔트로피에 대한 중요성은 다음과 같은 연구 결과를 통해 확인된다. 리눅스 의사난수발생기(Pseudo Random Number Generator, PRNG)의 입력 엔트로피가 충분하지 않아 일부 TLS/SSH 호스트(host)의 RSA/DSA 개인키를 정당하지 않은 사용자가 얻을 수 있다[3]. 부팅 시 충분하지 않은 엔트로피를 사용하는 안드로이드(Android) 난수발생기의 취약성을 이용하여 IPv6 서비스 거부 공격(denial of service attack)과 스택 카나리(stack canary) 우회가 성공하였다[4]. 또한, 난수발생기 내에 있는 엔트로피 수집기의 취약성 또는 엔트로피를 과대추정하거나 보수적으로 추정함으로 인한 공격들이 있다[5-8]. 따라서 퓨시아 및 지르콘에서 사용하는 난수발생기, 특히 입력인 잡음원에 대한 안전성 분석이 필요하다.

본 논문은 퓨시아 및 지르콘의 난수발생기에 대한 안전성 분석을 진행하였다. 본 논문은 2019년 3월까지 개발된 퓨시아 및 지르콘의 소스 코드(source code)[9]를 사용하여 분석 및 실험을 진행하였다. 지르콘 난수발생기가 사용하는 엔트로피 소스 중 지터(jitter) 엔트로피 소스는 퓨시아 및 지르콘의 모든 동작 환경에서 난수발생기에 엔트로피를 주는 유일한 요소가 될 가능성이 크다. 본 논문에서는 지터 엔트로피 소스에 대한 안전성 분석을 진행하였고, 지르콘이 지터 엔트로피 소스의 엔트로피를 과대 추정함을 확인하였다. 또한, 충분하지 않은 엔트로피를 갖는 지터 엔트로피 소스에 대한 전수조사량 계산 시나리오를 제시하였다. 지터 엔트로피 소스는 퓨시아 및 지르콘뿐만 아니라 Linux 등 여러 운영체제에서 사용되고 있다. 또한, 국내 표준문서[10]에서도 여러 운영체제에서 지터가 엔트로피 소스로 사용될 수 있다고 제시하고 있다. 따라서 본 연구 결과는 다양한 환경에서 수집되는 지터 엔트로피 소스에 대한 공격량을 계산하는 데 활용되고, 지터 엔트로피 소스가 사용되는 여러 운영체제의 난수발생기에 대한 안전성 분석에 사용될 것으로 기대한다.

2장에서는 퓨시아 및 지르콘에서 사용하는 난수발생기의 구조와 사용되는 세 가지의 엔트로피 소스를 소개한다. 3장에서는 난수발생기의 주된 입력으로 사용될 가능성이 가장 큰 지터 엔트로피 소스에 대한 안전성 분석을 진행한다. 4장에서는 본 논문의 결론을 맺는다.

1.1 용어

본 논문에서 사용되는 몇 가지 용어를 정리하면 다음과 같다.

- 엔트로피 소스(entropy source) : 잡음원과 동일한 의미로 사용
- 샘플(sample) : (디지털화된) 잡음원의 1회 출력으로 얻은 데이터
- 샘플 크기 : 비트(bit) 단위의 샘플 크기
- 엔트로피(entropy) : 관찰 이전의 실험 결과에 대한 지식과 관련된 정의로, 값을 예측하는 것에 대한 불확실성을 반영함[11]. 엔트로피가 클수록 예측에 대한 불확실성이 커짐. 확률질량함수(probability mass function)가 p인 확률변수(random variable) X의 엔트로피 $H(X) = -\sum_{\mathrm{x}} p(\mathrm{x}) log_2 p(\mathrm{x})$
- 최소 엔트로피(min-entropy) : 예측 불가능성의 척도 중 가장 작은 값을 갖는 엔트로피[12]. 확률질량

함수가 p인 확률변수 X의 최소 엔트로피 $H_{\infty}(X) = \min_{x}(-log_2 p(x))$. 이때, $H_{\infty}(X) \leq H(X)$이며, 균등분포일 때 등호가 성립함

II. 퓨시아 및 지르콘 난수발생기

2.1 퓨시아 및 지르콘 난수발생기의 구조 및 특징

퓨시아 및 지르콘의 난수발생기(Cryptographically secure Pseudo Random Number Generator, CPRNG)는 논블록킹(non-blocking) 방식으로 암호학적으로 안전한 의사난수 데이터를 제공한다. 이 난수발생기를 Zircon CPRNG라 명하겠다. Fig.1.에서 보이는 바와 같이 Zircon CPRNG는 스트림(stream) 암호 알고리즘인 ChaCha20을 의사난수 출력함수로 사용한다. ChaCha20의 입/출력 데이터의 크기는 128 비트이고 키(key)의 크기는 256 비트이다. 키는 난수발생기의 시드(seed) 역할을 한다.

사용자는 시스템 호출(system call)을 통해 Zircon CPRNG에 접근하여 난수를 출력하거나 키를 갱신할 수 있다. 각 호출에 따른 난수발생기의 동작 과정은 다음과 같다.

◆ zx_cprng_draw()을 통한 난수 출력

: Zircon CPRNG는 사용자로부터 받은 데이터를 ChaCha20의 입력으로 사용하고, 카운터 모드(counter mode)를 통해 사용자가 요청한 길이(최대 2^{38} 바이트)의 난수를 출력함

◆ zx_cprng_add_entropy()를 통한 키 갱신

: Zircon CPRNG는 사용자로부터 받은 데이터와 기존 키를 연접하여 SHA256의 입력으로 사용하고, 출력인 해시값으로 키를 갱신함

지르콘 커널이 부팅될 때 난수발생기의 생성 및 동작 과정은 다음과 같다.

1) 부팅 초기에 Zircon CPRNG를 인스턴스화(instantiation)하고, 이를 GlobalPRNG로 명한다. ChaCha20의 입력 데이터와 키를 0으로 초기화한다.

2) 커널이 동작하고 있는 환경에서 사용할 수 있는 엔트로피 소스를 확인하고, 256 비트의 엔트로피만큼 엔트로피 소스의 샘플들을 수집한다. 수집한 엔트로피 소스와 기존 키를 연접하여 SHA256의 입력으로 사용하고, 출력되는 해시값으로 키를 갱신한다. 키는 사용할 수 있는 엔트로피 소스의 개수만큼 갱신된다.

3) 커널에서 동작하는 각 프로세스(process)는 Zir-con CPRNG를 인스턴스화한다. 그런 다음, GlobalPRNG의 출력 난수와 0으로 초기화된 키를 입력으로 하는 SHA256의 출력값으로 키를 갱신한다. 이때, 사용된 GlobalPRNG의 출력 난수는 재사용되지 않는다.

본 논문이 분석 및 실험에 사용한 소스 코드 내 GlobalPRNG의 키는 부팅 초기 갱신된 이후 추가 갱신되지 않는다. 2020년 12월 버전의 소스 코드에

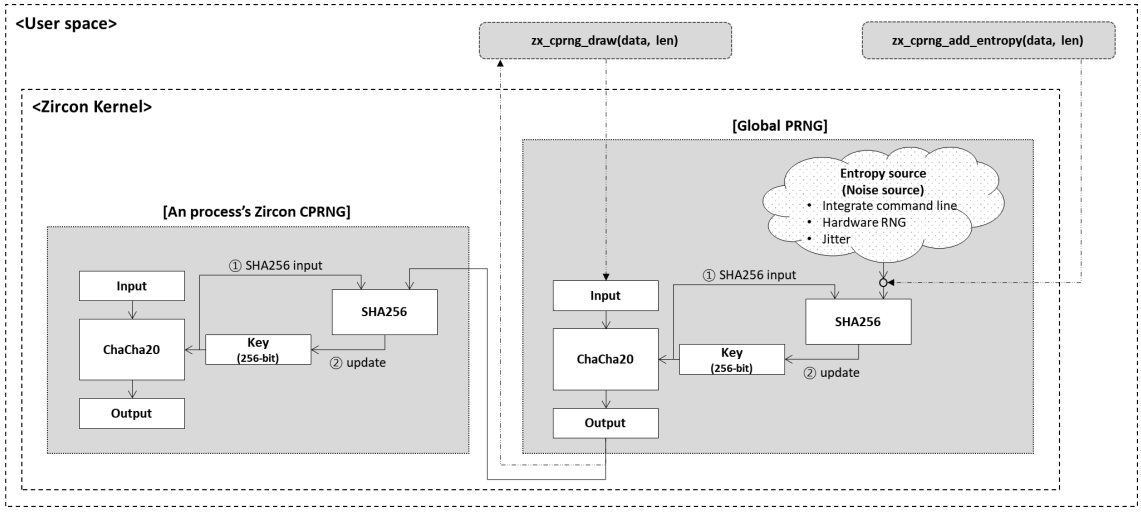

Fig. 1. The architecture of the random number generator in Zircon kernel

서는 30초마다 하드웨어 난수발생기 또는 지터 엔트로피 소스를 수집하여 GlobalPRNG의 키를 갱신하는 스레드가 생성된다.

2.2 퓨시아 및 지르콘 난수발생기의 엔트로피 소스

Zircon CPRNG는 커널 내에서 직접 접근할 수 있는 엔트로피 소스만 사용한다. 통합 명령어(integrate command line), 하드웨어 난수발생기, 지터가 엔트로피 소스로 사용될 수 있다. 각 엔트로피 소스에 대한 설명은 다음과 같다.

2.2.1 통합 명령어

리눅스 커널의 난수발생기인 /dev/urandom에서 32바이트 크기의 난수를 받아온 후, 이를 입력으로 한 SHA256의 출력 해시값이 하나의 샘플값이다. 샘플 크기는 256비트이고, 샘플당 256비트의 엔트로피를 가진다고 간주한다[13].

2.2.2 하드웨어 난수발생기

Intel의 RdRand, Amlogic의 하드웨어 난수발생기 등이 있다. 지르콘은 n비트 크기의 하드웨어 난수발생기의 출력 데이터가 n비트의 엔트로피를 가진다고 간주한다[13].

2.2.3 지터

CPU 명령어 처리 시간(instruction timing)을 엔트로피 소스로 사용한다. Müller, S.가 제안한 CPU jitter NPTRNG(Non-Physical True Random Number Generator)[14] 또는 일부 구조(이후, raw data)로 구성되어 있다. CPU jitter NPTRNG의 출력 데이터의 샘플 크기는 1비트이고, raw data의 샘플 크기는 8비트이다. Zircon CPRNG는 기본적으로 raw data를 사용한다. 사용자가 원하면 명령어 kernel.jitterentropy.raw의 값을 0(false)으로 설정하여 엔트로피 소스를 CPU jitter NPTRNG로 변경할 수 있다. 지르콘은 1,000바이트 크기의 엔트로피 소스 데이터가 450비트의 엔트로피를 가진다고 간주한다[13]. 2020년 12월 버전의 소스 코드에서는 50비트의 엔트로피를 가진다고 수정되었다[9].

2.2.3.1 CPU jitter NPTRNG

Fig.2.에서 보이는 바와 같이 CPU jitter NPTRNG[14]는 초기화 과정을 진행하고 64번의 라운드를 진행하여 64비트 크기의 난수를 출력하는 난수발생기이다. 또한, 64비트 크기의 풀(pool)을 가지고 있다. 한 라운드는 네 가지의 단계로 구성되어 있다. 이는 메모리 접근(memory access), 델타(delta) 계산, LFSR(Linear Feedback Shift Register), 정적 테스트(stuck test)이다. 각 라운드의 델타 계산 단계에서 계산된 64비트 크기의 current_delta 값을 이용하여 풀을 갱신한다. current_delta는 CPU jitter NPTRNG이 사용하는 잡음원이다.

하나의 라운드를 구성하는 네 가지 단계에 대한 각각의 설명은 다음과 같다.

(1) 메모리 접근 단계

(블록의 바이트 단위 크기 × 블록 개수) 크기의 메모리에 cnt_m번 접근한다. 접근 방법은 (블록 크기

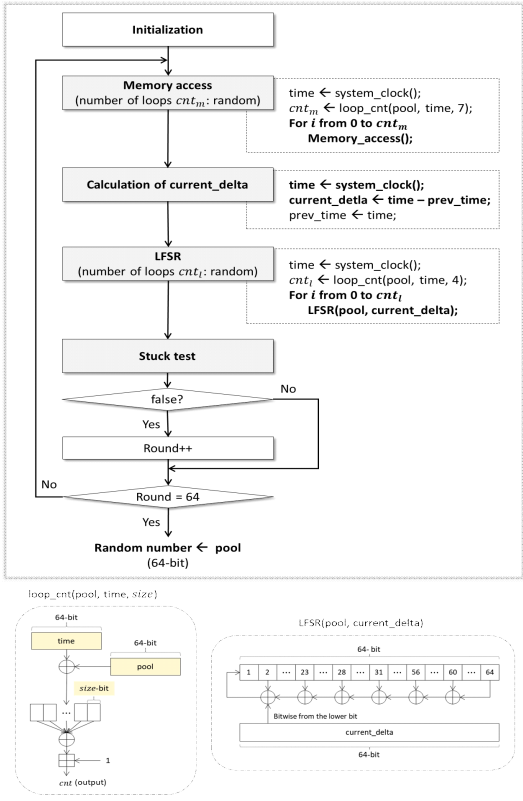

Fig. 2. The structure of CPU jitter NPTRNG

- 1) 단위로 움직이면서 해당 자리의 바이트 값과 1에 대해 모듈러-256(modular-256) 덧셈 연산을 하는 것이다. 이를 통해 메모리 접근 단계의 소요 시간, 즉 current_delta에 변동성을 준다. 지르콘은 메모리 블록의 크기를 64 바이트로 설정하고 블록의 개수를 512 개로 설정하여 32 KB 크기의 메모리를 사용한다. 메모리 접근 횟수 cnt_m은 라운드마다 129 ~ 256 범위의 랜덤한 값으로 선택된다. 선택 방법은 다음과 같다. 메모리 접근 단계가 시작될 때마다 시스템 클록(system clock)으로 측정한 64 비트 크기의 타임스탬프(timestamp)와 풀의 값에 대해 XOR 연산을 수행한다. 그런 다음, 이를 7 비트 크기의 블록으로 나누고, 나누어진 모든 블록에 대해 XOR 연산을 수행한 후 1을 더한 값이 cnt_m이 된다.

(2) 델타 계산 단계

풀을 업데이트하는 요소인 current_delta를 계산한다. 매 라운드 델타 계산 단계가 시작될 때 시스템 클록으로 타임스탬프를 측정한다. 그런 다음, 이전 라운드에서 측정한 타임스템프와의 차이를 계산한 값이 current_delta가 된다. 따라서 current_delta는 이전 라운드의 델타 계산, LFSR과 정적 테스트 단계의 소요 시간과 현재 라운드의 메모리 접근 단계의 소요 시간의 합이다.

(3) LFSR 단계

델타 계산 단계에서 계산된 current_delta를 비트 단위로 원시 다항식이 $x^{64}+x^{61}+x^{56}+x^{31}+x^{28}+x^{23}+1$인 LFSR에 적용하여 풀을 업데이트한다. 이때, 이 과정은 공회전 횟수 cnt_l만큼 반복 진행된다. 공회전을 통해 LFSR 단계의 소요 시간에 변동성을 주며, 공회전 횟수와 관계없이 업데이트된 풀의 결과는 같다. 라운드마다 cnt_l은 1 ~ 16 범위의 랜덤한 값으로 선택된다. 메모리 접근 횟수 cnt_m을 계산하는 방법에서 7 비트 대신 4 비트로 변경하여 계산한 값이 cnt_l이 된다.

(4) 정적 테스트 단계

current_delta가 고정된 상태인지 확인한다. current_delta, 2차 차분 값과 3차 차분 값 중 하나라도 0이 나오면 current_delta가 고정된 상태라고 판정한다. 이 경우, 해당 라운드를 전체 실행된 라운드 횟수에 추가하지 않는다. 고정된 상태가 아니라면, 풀을 7 비트만큼 왼쪽으로 회전시킨다. 차분 값은 이전 라운드들의 current_delta를 이용하여 계산한다.

2.2.3.2 Raw data

Raw data는 CPU jitter NPTRNG의 일부 구조를 사용한다. Fig.3.에서 보이는 바와 같이 CPU jitter NPTRNG의 메모리 접근 단계와 LFSR 단계에 대한 소요 시간을 시스템 클록으로 측정하고, 소요 시간의 하위 8 비트를 하나의 샘플로 생성한다. 지르콘은 메모리 접근 단계의 cnt_m는 160 번, LFSR 단계의 cnt_l는 1 번으로 고정한다.

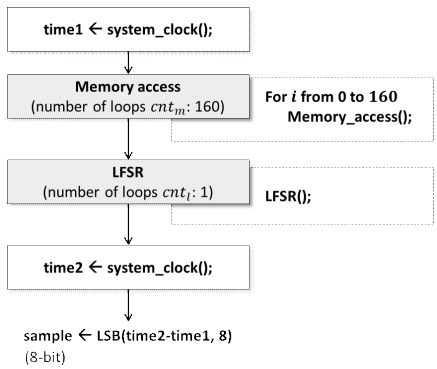

Fig. 3. The structure of raw data

III. 지터 엔트로피 소스의 안전성 분석

2.1절에서 살펴본 바와 같이, 지르콘은 부팅 초기에 GlobalPRNG를 생성한다. 커널 내 각 프로세스는 Zircon CPRNG를 생성하여 GlobalPRNG의 출력 난수를 엔트로피 소스로 사용한다. 또한, 응용프로그램에서 사용자는 GlobalPRNG의 출력 난수를 사용한다. 지르콘 부팅 초기 엔트로피 소스의 엔트로피가 충분하지 않아 엔트로피 소스 데이터를 예측할 수 있으면, 결정론적 알고리즘으로 구성된 GlobalPRNG의 출력 난수를 알 수 있다. 또한, 각 프로세스 난수발생기의 출력 난수와 사용자에게 전달한 출력 난수도 알 수 있다. 따라서 엔트로피 소스의 취약성이 응용프로그램에 대한 공격까지 이루어질 가능성이 있다.

지르콘의 난수발생기는 2.2절에서 살펴본 바와 같이 세 가지의 엔트로피 소스를 사용한다. 지르콘이 리눅스 커널이 없는 환경에서 동작할 경우, 통합 명

령어를 사용할 수 없다. 또한, IoT 환경과 같은 경량 환경에서 통합 명령어와 하드웨어 난수발생기는 사용되지 못할 가능성이 크다. 반면, 지터는 모든 지르콘의 동작 환경에서 GlobalPRNG에 엔트로피를 줄 수 있다. 특히, 경량 환경에서는 유일하게 엔트로피를 줄 가능성이 크다. 따라서 본 장에서는 지터 엔트로피 소스에 대한 안전성을 분석하고자 한다. 지르콘은 지터 엔트로피 소스로 raw data를 사용하므로, 이에 대한 안전성을 분석하고자 한다.

3.1 지터 엔트로피 소스의 엔트로피 분석

지르콘은 1,000 바이트 크기의 지터 엔트로피 소스 데이터가 450 비트의 엔트로피를 가진다고 간주한다[13]. Raw data의 샘플 크기는 8 비트이므로, 샘플당 0.45 비트의 엔트로피를 가진다는 것이다. 본 절에서는 지르콘이 동작하는 환경에서 지터 엔트로피 소스의 엔트로피를 추정하고, 지르콘의 엔트로피 추정치에 대한 타당성을 검증하고자 한다.

퓨시아와 지르콘은 데스크톱부터 IoT 환경에서도 사용될 수 있도록 개발되고 있다. 따라서 본 논문에서는 데스크톱과 IoT 기기의 중간 사양에 해당하는 두 가지의 보드를 실험 환경으로 설정하였다. 각 보드에 대한 사양은 Table 1.에 나타나 있다.

실험은 다음과 같이 진행하였다. 각 보드에서 커널 부팅 초기에 백만 개의 샘플을 수집하였다. 대표적인 문서 NIST SP 800-90B(2nd draft)[15]의 엔트로피 추정법을 이용하여 수집한 엔트로피 소스의 엔트로피를 추정하였다. 2018년 1월, SP 800-90B 최종 버전[11]이 공표되었으나, 구글이 지르콘의 안전성을 분석할 때 2nd draft 문서를 사용하였으므로 본 실험에서도 이 문서를 사용하였다. 부팅 초기 샘플 수집과 엔트로피 추정을 20 번 반복 진행하였다.

Table 1. Configurations of experimental platforms

Board	HiKey 960	Khadas VIM2
SoC	Kirin 960	Amlogic S912
Processor	4 Cortex A73 + 4 Cortex A53 Big.Little CPU architecture	64 bit Octa core Cortex A53
RAM	3 GB LPDDR4	3 GB DDR4
Application	Development board	TV box, mini PC, nano server

Table 2. Min-entropy of jitter entropy source

Board	HiKey 960	Khadas VIM2
Average of min-entropy	0.12	0.37
Maximum of deviation	0.014	0.017

(min-entropy per sample(8-bit))

각 보드에서 수집한 지터 엔트로피 소스의 최소 엔트로피는 Table 2.에 작성되어 있다. HiKey 960에서 수집한 지터 엔트로피 소스의 엔트로피는 샘플당 0.12 비트로 추정되었고, Khadas VIM2에서는 샘플당 0.37 비트로 추정되었다. 지르콘은 샘플당 0.45 비트로 추정하고 있으므로, 실험을 통해 지르콘이 엔트로피를 과대 추정하고 있음을 확인하였다. 지르콘은 부팅 초기에 569 개의 지터 샘플을 수집하여 풀 엔트로피(full-entropy)를 갖는 ChaCha20의 256 비트의 키를 생성한다. 왜냐하면 아래와 같은 식에 따라 569 개의 지터 샘플에 대한 이론적 예측 가능성의 한계치가 256 비트이기 때문이다.

샘플당 0.45 비트의 엔트로피 × 569 개의 샘플
= 256 비트의 엔트로피

지르콘은 엔트로피를 과대 추정하고 있으므로, 지르콘이 동작하는 환경에서 수집한 지터 엔트로피 소스의 이론적 예측 가능성의 한계지는 256 비트보다 적다. HiKey 960의 경우 569 개의 샘플에 대한 이론적 예측 가능성의 한계치가 68 비트이고, Khadas VIM2의 경우 210 비트이다. 따라서 지르콘이 주장한 256 비트 계산량보다 적은 계산량으로 569 바이트 크기의 지터 엔트로피 소스를 예측할 수 있다. 또한, 이로부터 GlobalPRNG의 출력 난수뿐만 아니라 각 프로세스의 출력 난수까지도 예측할 가능성이 있다.

3.2 지터 엔트로피 소스의 예측 가능성 분석

3.1절에서 지터 엔트로피 소스의 이론적 예측 가능성의 한계치를 지르콘이 과대 추정하고 있음을 보였다. 본 절에서는 지터 엔트로피 소스에 대한 실제 예측 가능성을 분석하고자 한다. 따라서 HiKey 960에서 수집한 569 바이트 크기의 지터 엔트로피 소스의 특성을 분석하고, 예측에 필요한 계산량을 분석한다.

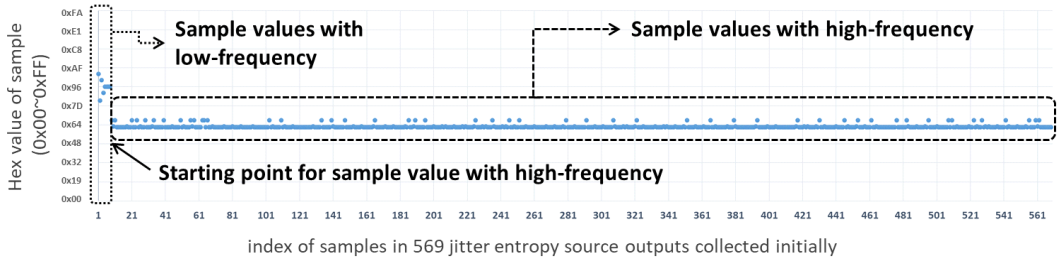

Fig. 4. Graph and defined terms for 569-byte jitter entropy source

3.2.1 지터 엔트로피 소스의 특성 분석

HiKey 960에서 지르콘 부팅 초기에 수집한 569개의 지터 샘플의 값의 변동성을 파악하기 위해 그래프로 도식하면 Fig.4.와 같다. 그래프의 x 축은 569개의 샘플에 대한 인덱스(index)이고, y 축은 크기가 8비트인 샘플의 헥사(Hex) 값이다. Fig.4.에서 보이는 바와 같이 HiKey 960에서 수집한 569개의 샘플은 특정한 바이트의 인덱스 이후부터 몇 개의 특정한 샘플값만 가진다. 반복 실험을 통해, 이 결과는 HiKey 960에서 수집한 지터 엔트로피 소스의 특성으로 확인하였다.

본 논문은 지터 엔트로피 소스의 특성을 반영하여 다음과 같이 세 가지의 용어를 정의하였다.

- 빈도수가 높은 샘플값
  : 특정 바이트 인덱스 이후부터 반복되는 몇 개의 특정한 샘플값
- 빈도수가 낮은 샘플값
  : 빈도수가 높은 샘플값이 아닌 샘플값
- 시작점(빈도수가 높은 샘플값의 시작점)
  : 빈도수가 높은 샘플값으로만 반복되기 시작하는 샘플의 인덱스

HiKey 960에서 지르콘 부팅 초기에 569개의 지터 샘플을 수집하여 각 용어에 해당하는 샘플값의 발생 빈도수를 확인하였다. Table 3.은 569개의 샘플에서 빈도수가 높은 샘플값의 발생 빈도수를 보여준다. 빈도수는 6,000번 반복 실험한 결과의 평균값이다. Table 3.에 작성된 바와 같이, 빈도수가 높은 샘플값은 0x61, 0x62와 0x6A였다. Fig.5.에서 보이는 바와 같이, 0x61인 샘플값의 발생 확률이 0.72로 가장 높았고, 나머지 각 샘플값의 발생 확률은 0.14이었다. Table 4.에는 6,000번의 반복 실험에서 발생하고 발생 확률이 0.03 이상인 빈도수가 낮은 샘플값과 빈도수가 작성되어 있다. 이는 569개의 샘플에서 빈도수가 낮은 샘플값의 발생 빈도수는 낮기 때문이다. 실험 결과, 빈도수가 낮은 샘플값에 대한 표본 공간의 크기는 50이었다. 또한, 부팅 초기에 수집한 569개의 지터 샘플에서 9번째 샘플(시작점)부터 빈도수가 높은 샘플값만 가질 가능성이 91%이고, 시작점이 10일 가능성이 7%이었다.

Table 3. Sample values (0x00 ~ 0xFF) with high-frequency and their frequency and probability

Sample value with high-frequency	0x61	0x62	0x6A
Average frequency	403	80	78

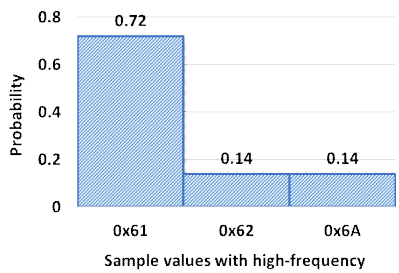

Fig. 5. Probability of sample values with high-frequency

Table 4. Sample values (0x00 ~ 0xFF) with low-frequency with probability over 0.03 and their probability

Sample value with low-frequency	0x96	0x9F	0x8D
Probability	0.37	0.18	0.11

Sample value with low-frequency	0x97	0x8E	0xA8
Probability	0.07	0.06	0.03

3.2.2 지터 엔트로피 소스의 예측 계산량 분석

569 바이트 크기의 지터 엔트로피 소스에 대한 일반적인 전수조사량을 계산하면 4,552 비트이다. 본 절에서는 지터 엔트로피 소스의 특성을 이용하여 4,552 비트의 전수조사량을 축소하고자 한다. 더 나아가 실험 분포와 최적 추측 공격(optimum guessing attack) 시나리오[11]를 이용하여 전수조사량을 축소하고자 한다.

3.2.2.1 지터 엔트로피 소스의 특성을 이용한 전수조사량 분석

3.2.1절에서 분석한 결과를 기반으로 569 바이트의 지터 엔트로피 소스에 대한 전수조사량을 다음과 같은 방법[16]으로 계산하였다. 먼저, 빈도수가 높은 샘플값의 시작점을 발생 확률이 0.91인 9 번째 샘플로 설정하였다. 9 ~ 569 번째 샘플의 값은 세 가지의 빈도수가 높은 샘플값 중 하나가 되므로, 빈도수가 높은 샘플값에 대한 확률 분포가 균등분포라는 가정에서 561 개의 샘플 에 대한 전수조사량을 3^{561}으로 계산하였다. 1 ~ 8 번째 샘플의 값은 50 개의 빈도수가 낮은 샘플값 중 하나가 되므로, 8 개의 샘플에 대한 전수조사량을 약 $(2^6)^8$으로 계산하였다. 따라서 569 바이트의 지터 엔트로피 소스에 대한 특성을 고려하였을 때 전수조사량은 $(2^6)^8 \times 3^{561} \approx 2^{938}$으로 계산되었다[16]. 이는 4,552 비트의 일반적인 전수조사량을 938 비트로 축소한 것이다.

3.2.2.2 실험 분포 및 최적 추측 공격 시나리오를 이용한 전수조사량 분석

앞서 전수조사량을 분석한 방법에서는 빈도수가 높은 샘플값에 대한 확률 분포가 균등분포라는 가정하에 계산하였다. 하지만 Fig.5.에 보이는 바와 같이 실제 확률 분포는 균등분포가 아님을 실험적으로 확인하였다. 따라서 본 절에서는 빈도수가 높은 샘플값에 대한 실험 분포(Fig.5.)를 활용하여 938 비트의 전수조사량에 대한 축소 가능성을 분석하고자 한다.

실험 분포를 활용하기 위해 전수조사량 분석에 최적 추측 공격 시나리오를 사용하였다. 최적 추측 공격에 관한 설명은 다음과 같다. 공격 대상인 샘플의 크기가 N 바이트라 하자. 공격 대상의 값을 확률변수 X라 하였을 때, 확률변수 X의 상태공간(state space)은 $S_X = \{x_1, x_2, \ldots, x_{(256^N)=L}\}$이고, 확률변수 X는 확률 $p_i = P(X = x_i)$, $1 \le i \le L$인 확률 분포 $p = (p_1, \ldots, p_L)$를 갖는다고 하자. 또한, 일반성을 잃지 않고 $p_1 \ge p_2 \ge \cdots \ge p_L$라 가정한다. 최적 추측 공격은 확률 분포 p에서 확률이 높은 순서부터 가져온 k 개의 확률만을 사용하여 다수의 공격 대상 중 하나에 대한 추측을 성공하는 공격이다. 따라서 임의의 $j \in \{1, 2, \ldots\}$ 번째 공격 대상인 샘플 X_j에 대해, 확률이 높은 순서부터 X_j가 x_i $(1 \le i \le k)$와 일치하는지 확인한다. 일치하면 공격에 성공한 것이므로 공격을 종료한다. 그렇지 않으면 임의의 $j+1$ 번째 공격 대상인 샘플 X_{j+1}에 대해 동일한 과정을 진행한다.

이러한 공격 방법으로부터 공격에 성공하는 평균 추측 횟수 $W_k(p)$는 식 (1)과 같이 계산된다[17]. 이때, $T^{(k)}$는 공격 성공까지 사용된 공격 대상의 개수를 의미하고, $G_t^{(k)}$는 t 번째 샘플에 대해 진행한 일치 여부 확인 횟수를 나타낸다[17]. 예를 들어, 첫 번째 공격 대상인 샘플 X_1이 모든 x_i $(1 \le i \le k)$와 일치하지 않고 두 번째 공격 대상인 샘플 X_2가 x_q $(1 \le q \le k)$와 일치한다고 가정하자. 이 경우, $T^{(k)} = 2$, $G_1^{(k)} = k$이고 $G_2^{(k)} = q$가 된다.

$$\begin{aligned} W_k(p) &= \mathrm{E}\left[\left(T^{(k)} - 1\right) \times k + G_{T^{(k)}}^{(k)}\right] \\ &= k\,\mathrm{E}\left[T^{(k)}\right] - k + \mathrm{E}\left[G_{T^{(k)}}^{(k)}\right]. \end{aligned} \tag{1}$$

$T^{(k)}$ 번째 공격 대상인 샘플 $X_{T^{(k)}}$에 대한 추측에 성공할 확률은 샘플 $X_{T^{(k)}}$가 x_i $(1 \le i \le k)$일 확률이므로 $\sum_{i=1}^{k} p_i$이고, $\mathrm{E}\left[T^{(k)}\right] = \frac{1}{\sum_{i=1}^{k} p_i}$이다. $G_{T^{(k)}}^{(k)}$가 j $(1 \le j \le k)$일 확률은 샘플 $X_{T^{(k)}}$가 추측에 성공하는 공격 대상일 때 $X_{T^{(k)}}$가 x_j일 확률이므로 $\frac{p_j}{\sum_{i=1}^{k} p_i}$이고, $\mathrm{E}\left[G_{T^{(k)}}^{(k)}\right] = \frac{\sum_{j=1}^{k} (j \times p_j)}{\sum_{i=1}^{k} p_i}$이다. 계산한 $\mathrm{E}\left[T^{(k)}\right]$와 $\mathrm{E}\left[G_{T^{(k)}}^{(k)}\right]$를 이용하여 식 (1)을 다시 작성하면 식 (2)[11]와 같다. 이때, $W_k(p)$를 최소화

하는 k^*가 선택된다.

$$W_k(p)=\frac{p_1+\cdots+(k-1)p_{k-1}+k\left(1-\sum_{i=1}^{k-1}p_i\right)}{\sum_{i=1}^{k}p_i}. \quad (2)$$

평균 추측 횟수 $W_{k^*}(p)$와 최소 엔트로피 $H_\infty(X)$ 간의 관계는 식 (3)[11]과 같다. 이때, 최소 엔트로피 $H_\infty(X)$는 $-\log_2(p_1)$이다.

$$H_\infty(X)\le W_{k^*}(p)\le H_\infty(X)+1. \quad (3)$$

최적 추측 공격 시나리오를 빈도수가 높은 샘플값으로만 구성된 엔트로피 소스에 적용하기 위해, 빈도수가 높은 샘플값 각각의 발생 횟수에 대한 분포를 도출하였다. 이를 위해 다음과 같은 실험을 진행하였다. 먼저 앞선 실험에서 사용한 6,000개의 엔트로피 소스 중에서 9번째 샘플이 시작점인 5,460개의 엔트로피 소스를 추출하였고, 이때의 각 엔트로피 소스의 크기는 569바이트이다. 추출된 각 엔트로피 소스에서 1 ~ 8번째 샘플을 제외하여 빈도수가 높은 샘플값만 갖도록 구성하였고, 이로부터 각 엔트로피 소스의 크기는 561바이트가 된다. 그런 다음, 각각의 엔트로피 소스에서 빈도수가 높은 샘플값 각각에 대한 빈도수를 측정하였다. 실험 결과, 빈도수가 높은 샘플값 각각에 대한 분포는 Fig.6.에 나타난 바와 같다. 또한, 각 샘플값에 대한 최대 빈도수와 최소 빈도수는 Table 5.에 작성되어 있다.

빈도수가 높은 샘플값에 대한 실험 분포와 각 샘플에 대한 실험 분포를 최적 추측 공격 시나리오에 적용하기 위해, 아래의 세 가지 가정을 하였다.

(1) 각각의 샘플(샘플 크기: 8비트)에서 빈도수가 높은 샘플값이 발생할 확률은 다음과 같다.

$$P(X=0\text{x}61)=0.72,$$
$$P(X=0\text{x}62)=0.14,$$
$$P(X=0\text{x}6\text{A})=0.14.$$

(2) 빈도수가 높은 샘플값 각각의 발생 횟수에 대한 범위는 다음과 같다. ($N_{0\text{x}61}+N_{0\text{x}62}+N_{0\text{x}6\text{A}}=561$)

$$357\le N_{0\text{x}61}\le 444,$$
$$61\le N_{0\text{x}62}\le 98,$$
$$36\le N_{0\text{x}6\text{A}}\le 123.$$

(3) 가정 (2)를 만족하지 않는 엔트로피 소스에 대한 확률은 0이다.

세 가지의 가정하에, 최적 추측 공격 시나리오를 빈도수가 높은 샘플값으로만 구성된 561바이트의 지터 엔트로피 소스에 적용하였다. 이때, 최적 추측 공격에 사용되는 공격 대상인 하나의 샘플은 561바이트의 지터 엔트로피 소스이다. 앞서 살펴본 식 (3)을 이용하였을 때, 561바이트 크기의 지터 엔트로피 소스에 대한 전수조사량인 평균 추측 횟수 $W_{k^*}(p)$는 543비트로 계산되었다. 이때, 식(2)에서 k의 값을 변경하여 $W_{k^*}(p)=2^{543}$이 되는 k^*의 값 ($\approx 1.4\times 10^{158}$)을 찾았다.

$$W_{k^*}(p)$$
$$=\frac{p_1+2p_2+\cdots+(k^*-1)p_{k^*-1}+k^*\left(1-\sum_{i=1}^{k^*-1}p_i\right)}{\sum_{i=1}^{k^*}p_i}$$
$$\approx 1.7\times 10^{163}\approx 2^{543}.\ \ (k^*\approx 1.4\times 10^{158})$$

따라서 569바이트의 지터 엔트로피 소스에 대한 전수조사량은 $(2^6)^8\times 2^{543}=2^{591}$으로 계산된다. 이를 다시 말하면, 569바이트의 지터 엔트로피 소스는

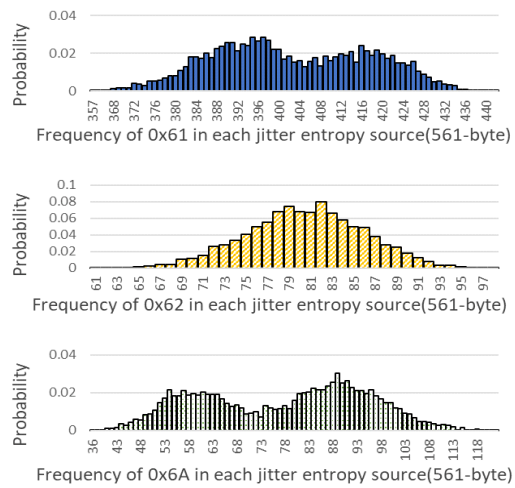

Fig. 6. Frequency distribution of each sample value with high-frequency in 5,460 jitter entropy sources (561-byte)

Table 5. Maximum and minimum frequencies for each sample value with high-frequency in in 5,460 jitter entropy sources (561-byte)

Sample value with high-frequency	0x61	0x62	0x6A
Maximum	444	98	123
Minimum	357	61	36

591 비트의 전수조사량으로 예측할 수 있다. Fig.7.에서 보이는 바와 같이, 실험 분포와 최적 추측 공격 시나리오를 이용하여 계산한 전수조사량은 일반적인 전수조사량인 4,552 비트보다 약 $2.4\times10^{1,192}$ 배 축소한 결과이다. 또한, 이는 균등분포라는 가정하에 지터 엔트로피의 특성을 이용하여 계산한 전수조사량인 938 비트보다 약 2.8×10^{104} 배 축소한 결과이다.

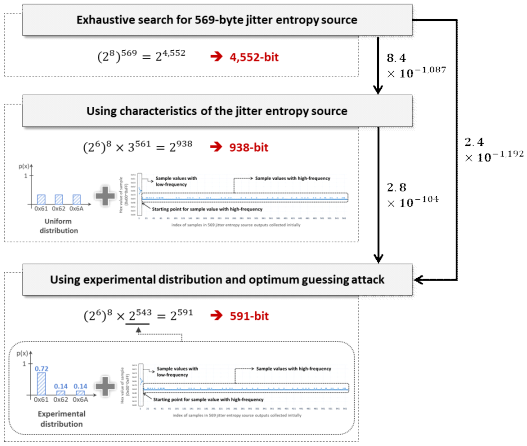

Fig. 7. The results of predictability analysis for 569-byte jitter entropy source

3.2.2.3 지터 엔트로피 소스에 대한 전수조사량 분석법 활용

지터 엔트로피 소스는 수집하는 환경에 따라 지터 엔트로피 소스의 특성(빈도수가 높은 샘플값의 개수, 샘플값에 대한 분포 등)이 달라진다. 따라서 본 절에서는 수집하는 샘플의 개수와 각 샘플값의 확률을 파라미터로 설정하고, HiKey 960과 Khadas VIM2 보다 열악한 환경에서 수집되는 지터 엔트로피 소스의 전수조사량을 계산하고자 한다.

IoT 기기와 같은 열악한 환경에서 수집되는 지터 엔트로피 소스를 분석 대상으로 설정하였으므로, 두 가지의 샘플값(d_1, d_2)으로만 지터 엔트로피 소스가 구성된다고 가정하자. 이때, $P(X=d_1)=p$이고, $p \geq 0.5$라 하자. 또한, 수집되는 샘플(샘플 크기: 8 비트)의 개수를 N이라 할 때, N개의 샘플 중에서 샘플값이 d_1인 샘플이 $N_1 = pN$개 있다고 가정하자. 이와 같은 가정하에서 N과 p의 값에 따른 평균 예측 횟수 $W_{k^*}(p)$의 값의 변화는 Fig.8.에 나타나 있다. 이때, 식 (3)에서 나타난 바와 같이, $W_{k^*}(p)$는 $H_\infty(X)$와 $H_\infty(X)+1$ 사이의 값을 가지며, $H_\infty(X) = -\log_2\left(p^{N_1}\times(1-p)^{(N-N_1)}\right)$이다. 열악한 환경에서 $p=0.99$인 샘플이 $N=250$개 수집되어 난수발생기의 입력으로 사용될 경우, Fig.8.에서 보이는 바와 같이, 250 바이트 크기의 엔트로피 소스에 대한 전수조사량은 20 비트로 계산된다.

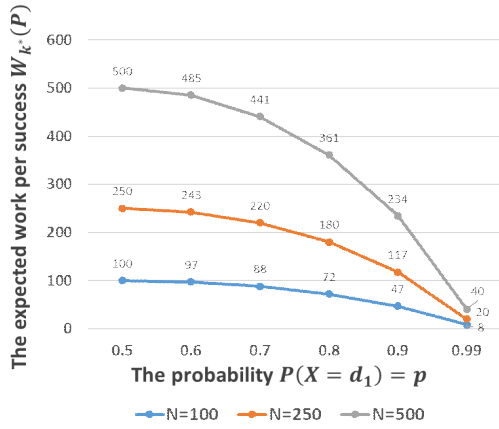

Fig. 8. The expected work per success $W_{k^*}(P)$ according to N and p

IV. 결 론

본 논문에서는 구글이 소스 코드를 공개하면서 개발하고 있는 퓨시아 및 지르콘 난수발생기의 구조 및 엔트로피 소스를 분석하였다. 또한, 퓨시아 및 지르콘이 동작할 수 있는 모든 환경에서 지르콘 난수발생기에 엔트로피를 주는 유일한 요소인 지터 엔트로피 소스에 대한 안전성 분석을 진행하였다. 그 결과, 지르콘이 지터 엔트로피 소스의 엔트로피를 과대 추정함을 확인하였다. 이로부터 HiKey 960에서 수집한 569 바이트 크기의 지터 엔트로피 소스에 대한 이론적 예측 가능성의 한계치가 지르콘의 주장인 256 비트가 아닌 68 비트임을 확인하였다. 그리고 지터 엔트

로피 소스에 대한 특성과 최적 추측 공격 시나리오를 이용하여 569 바이트 크기의 지터 엔트로피 소스에 대한 실제 전수조사량을 계산하는 시나리오를 제시하고, 전수조사량이 4,552 비트에서 591 비트로 축소될 수 있음을 보였다. 또한, 제시한 전수조사량 계산 시나리오의 활용으로 본 논문에서 사용한 실험 환경보다 열악한 환경에서 수집되는 지터 엔트로피 소스에 대한 전수조사량을 보였다. 본 논문에서 제시한 지터 엔트로피 소스에 대한 안전성 분석 과정은 지터를 엔트로피 소스로 사용하는 다른 운영체제의 난수발생기에 대한 안전성 분석에 사용될 것으로 기대한다.

References

[1] Fuchsia, "Fuchsia" https://fuchsia.dev/, last accessed 2020/12/28.

[2] Google Open Source, "Expanding Fuchsia's open source model" https://opensource.googleblog.com/2020/12/expanding-fuchsias-open-source-model.html, last accessed 2020/12/28.

[3] Heninger, Nadia, et al., "Mining your Ps and Qs: Detection of widespread weak keys in network devices," Proceedings of the 21st USENIX Security Symposium, pp. 205-220, Aug. 2012.

[4] Kaplan, David, et al., "Attacking the Linux PRNG On Android: Weaknesses in Seeding of Entropic Pools and Low Boot-Time Entropy," Proceedings of the 8th USENIX Workshop on Offensive Technologies, pp. 1-14, Aug. 2014.

[5] Yilek, Scott, et al., "When private keys are public: Results from the 2008 Debian OpenSSL vulnerability," Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement, pp. 15-27, Nov. 2009.

[6] Soo Hyeon Kim, Daewan Han, and Dong Hoon Lee, "Predictability of Android OpenSSL's pseudo random number generator," Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 659-668, Nov. 2013.

[7] Ding, Yu, et al., "Android low entropy demystified," Proceedings of the 2014 IEEE International Conference on Communications, pp. 659-664, Jun. 2014.

[8] Taeill Yoo, Ju-Sung Kang, and Yongjin Yeom, "Recoverable random numbers in an internet of things operating system," Entropy, 19(3), 113, pp. 1-14, Mar. 2017.

[9] Google Git, "fuchsia Git repositories" https://fuchsia.googlesource.com/, last accessed 2020/12/28.

[10] Choi, H., et al., "Guideline for the Collection and Application of Noise Source on Operating Systems," TTAS.KO-12.0235/R2, TTA, Dec. 2020.

[11] Turan, M. S., Barker, E., Kelsey, J., McKay, K., Baish, M., and Boyle, M., "Recommendation for the Entropy Sources Used for Random Bit Generation," NIST Special Publication 800-90B, NIST, Jan. 2018.

[12] Kim, H., et al., "Guideline for Testing Noise Sources used in Software Cryptographic Modules," TTAK.KO-12.0341/R1, TTA, Dec. 2020.

[13] Yewon Kim, Ju-Sung Kang and Yongjin Yeom, "Structure Analysis of Random Number Generator in Google's RTOS," Proceedings of the KICS summer conference 2018, pp. 1-2, Jun. 2018.

[14] Stephan Müller, "CPU Time Jitter Based Non-Physical True Random Number Generator," https://www.chronox.de/jent/doc/CPU-Jitter-NPTRNG.html, 2013.

[15] Sönmez Turan, M., Barker, E., Kelsey, J., McKay, K., Baish, M., and Boyle, M., "Recommendation for the Entropy Sources Used for Random Bit Generation(Second DRAFT)," NIST Special Publication 800-90B, NIST, Jan. 2016.

[16] Yewon Kim, Wontae Kim, Ju-Sung Kang, and Yongjin Yeom, "A Security Analysis of the Jitter Entropy Source of a Random Number Generator used in Zircon Operating System by Google," Proceedings of the KICS winter conference 2019, pp. 1-2, Jan. 2019.

[17] Ju-Sung Kang, Hojoong Park, and Yongjin Yeom, "Probabilistic Analysis for the Relationship Between Min-Entropy and Guessing Attack," Proceedings of the the 11th KIPS International Conference on Ubiquitous Information Technologies and Applications (CUTE 2016), pp. 567-572, Dec. 2016.

〈저자소개〉

김 예 원 (Yewon Kim) 학생회원
2015년 8월: 숙명여자대학교 수학과 학사
2017년 8월: 국민대학교 일반대학원 금융정보보안학과 석사
2017년 9월~현재: 국민대학교 일반대학원 금융정보보안학과 박사과정
〈관심분야〉 암호구현, 난수성 분석 및 평가, 병렬 프로그래밍

염 용 진 (Yongjin Yeom) 종신회원
1991년 2월: 서울대학교 수학과 졸업
1994년 2월: 서울대학교 수학과 석사
1999년 2월: 서울대학교 수학과 박사
2000년 4월~2012년 2월: ETRI 부설연구소 책임연구원/팀장
2012년 3월~현재: 국민대학교 과학기술대학 정보보안암호수학과 정교수
2013년~현재: 국민대학교 BK21+ 미래 금융정보보안 인력양성사업단 교수
〈관심분야〉 암호구현 및 분석, 보안시스템 평가

강 주 성 (Ju-Sung Kang) 종신회원
1989년 2월: 고려대학교 수학과 졸업
1991년 2월: 고려대학교 일반대학원 수학과 석사
1996년 2월: 고려대학교 일반대학원 수학과 박사
1997년~2004년: 한국전자통신연구원 선임연구원/팀장
2004년 3월~현재: 국민대학교 과학기술대학 정보보안암호수학과 정교수
2013년~현재: 국민대학교 BK21+ 미래 금융정보보안 인력양성사업단 교수
〈관심분야〉 암호이론, 정보보안 프로토콜, 안전성 분석 및 평가