KIPS Trans. Comp. and Comm. Sys.
Vol.12, No.2 pp.69~76
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2023.12.2.69

PDF Version 1.4-1.6 Password Cracking in CUDA GPU Environment

Hyun Jun Kim† · Si Woo Eum†† · Hwa Jeong Seo†††

ABSTRACT

Hundreds of thousands of passwords are lost or forgotten every year, making the necessary information unavailable to legitimate owners or authorized law enforcement personnel. In order to recover such a password, a tool for password cracking is required. Using GPUs instead of CPUs for password cracking can quickly process the large amount of computation required during the recovery process. This paper optimizes on GPUs using CUDA, with a focus on decryption of the currently most popular PDF 1.4-1.6 version. Techniques such as eliminating unnecessary operations of the MD5 algorithm, implementing 32-bit word integration of the RC4 algorithm, and using shared memory were used. In addition, autotune techniques were used to search for the number of blocks and threads that affect performance improvement. As a result, we showed throughput of 31,460 kp/s (kilo passwords per second) and 66,351 kp/s at block size 65,536, thread size 96 in RTX 3060, RTX 3090 environments, and improved throughput by 22.5% and 15.2%, respectively, compared to the cracking tool hashcat that achieves the highest throughput.

Keywords : Password Cracking, CUDA, Optimization, PDF

PDF 버전 1.4-1.6의 CUDA GPU 환경에서 암호 해독 최적 구현

김 현 준† · 엄 시 우†† · 서 화 정†††

요 약

매년 수십만 개의 암호를 분실하거나 잊어버리면서 합법적인 소유자나 권한을 부여받은 법 집행 담당자가 필요한 정보를 사용할 수 없게 된다. 이러한 암호를 되찾기 위해 암호 해독(Password Cracking)이 사용된다. 암호 해독에 CPU 대신 GPU를 사용하면 복구 과정에서 필요한 많은 양의 계산을 신속하게 처리할 수 있다. 본 논문은 현재 가장 많이 사용되는 PDF 1.4 -1.6 버전의 암호 해독에 중점을 두고 CUDA를 사용하여 GPU에서 최적화한다. MD5 알고리즘의 불필요 연산 제거, RC4 알고리즘의 32비트 워드 통합 구현, 공유메모리 사용의 기법을 사용하였다. 또한 성능향상에 영향을 미치는 블록, 스레드 수 탐색을 위해 오토튠 기법을 사용하였다. 결과적으로 RTX 3060, RTX 3090 환경에서 블록 크기 65,536, 스레드 크기 96에서 31,460 kp/s(kilo passwords per second), 66,351 kp/s의 처리량을 보였으며, 기존 최고 처리량을 보여주는 해시캣의 처리량보다 각각 22.5%, 15.2%를 향상시켰다.

키워드 : 패스워드 해독, CUDA, 최적화, PDF

1. 서 론

매년 수십만 개의 암호를 분실하거나 잊어버리면서 합법적인 소유자나 권한을 부여받은 법 집행 담당자가 필요한 정보를 사용할 수 없게 된다. 이러한 암호를 되찾기 위해 암호 해독(Password Cracking)이 사용된다. 암호 해독은 컴퓨터 시스템에 저장되거나 전송된 데이터에서 암호를 복구하는 작업을 말한다. 암호 해독으로 사용자는 잊어버린 암호를 복구하거나, 판사가 접근을 허용한 디지털 증거에 액세스할 수 있다. 악의적인 공격자는 무단으로 시스템의 접근 권한을 얻기 위해 사용한다. 암호 해독에서 암호를 알아내기 위해 많은 수의 문자, 숫자를 조합하여 대입한다. 일반적인 방법으로 사전 공격(Dictionary Attack)과 무차별 암호 대입 공격[1]이 있다. 사전공격은 자주 사용되거나 예측되는 사용자 암호를 대입하는 공격이다. 그러나 사전에 저장되지 않는 경우 모든 암호에 대하여 대입을 시도하는 무차별 암호 대입 공격을 해야 한다. 무차별 암호 대입 공격의 원리는 단순하나 많은 양의

※ 이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00540, GPU/ASIC 기반 암호알고리즘 고속화 설계 및 구현 기술개발, 80%) 그리고 이 성과는 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2020R1F1A1048478, 20%).
※ 이 논문은 2022년 한국정보처리학회 ASK 2022의 우수논문으로 "CUDA GPU 상의 PDF 1.4-1.6 해독 최적 구현"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 한성대학교 정보컴퓨터공학과 박사과정
†† 준 회 원 : 한성대학교 IT융합공학부 석사과정
††† 종신회원 : 한성대학교 IT융합공학부 조교수
Manuscript Received : August 1, 2022
First Revision : September 13, 2022
Second Revision : October 11, 2022
Accepted : October 22, 2022
* Corresponding Author : Hwa Jeong Seo(hwajeong84@gmail.com)

계산이 필요하다. 사용자의 암호가 길어질수록 더 많은 대입 과정이 요구되며 심각한 경우 수십억 개의 후보 암호를 확인해야 하며 해결하는 데 며칠에서 몇 달이 걸릴 수 있다. 많은 양의 계산을 신속하기 처리하기 위해서 암호 해독에 CPU 대신 GPU를 사용하는 연구가 활발하게 진행되고 있다[2][3]. 대부분의 최신 컴퓨터는 GPU를 탑재하며 병렬 처리를 지원하기 때문에 수많은 암호를 대입하여 처리해야 하는 암호 크래킹에 적합하다.

본 논문은 주로 사용되는 PDF 문서에 대한 암호 해독에 중점을 두었다. PDF(Portable Document Format)[4]는 Adobe에서 개발한 파일 형식이다. 소프트웨어, 하드웨어 및 운영체제와 무관하게 텍스트 서식 및 이미지를 포함한 문서를 사용할 수 있다. 이러한 범용성으로 모든 버전이 ISO 32000으로 표준화되었으며 전 세계적인 표준문서로 자리 잡았다.

일반적으로 암호화된 파일은 사용자가 설정한 비밀번호를 고유한 알고리즘으로 해싱하여 생성된 해시값을 내부에 포함한다. 이 해시값은 암호화된 파일의 내용을 복호화하기 전에 입력받은 비밀번호의 정확성을 검사할 때 사용된다. 따라서 암호화된 파일 암호 해독과정은 전체 파일의 복호화를 시도할 필요 없이 추측 암호를 고유한 알고리즘으로 해싱하여 저장된 해시값과 반복적으로 비교한다. PDF에는 이러한 해시를 생성하는 고유한 알고리즘이 공개되어 있으며 PDF의 버전에 따라 다른 해시 생성 알고리즘이 사용된다. 본 논문은 여러 PDF 버전 중 현재 가장 많이 사용되는 PDF 1.4 -1.6 버전의 암호 해독 알고리즘을 대상으로 대량의 암호 해독을 위한 GPU 상에서 속도 최적화를 진행하였다.

PDF 1.4 -1.6 버전의 암호 해독 알고리즘에서 많은 연산 시간을 차지하는 부분은 반복적으로 사용되는 MD5 알고리즘과 RC4 알고리즘이다. 따라서 본 논문은 MD5 알고리즘과 RC4 알고리즘을 중점적으로 최적화를 적용하였다. MD5에는 불필요 연산 제거, RC4는 32비트 워드 통합, 공유메모리 사용의 기법을 사용하였다. 그리고 GPU에서 성능에 영향을 미치는 요소인 블록, 스레드 수 탐색을 위해 오토튜 기법을 사용하였다. 결과적으로 제안 기법의 구현은 RTX 3060, RTX 3090 환경에서 그리드 당 블록 수는 65,536개, 블록당 스레드 수는 96개에서 가장 높은 성능을 달성하였고 각각 31,460 kp/s(kilo passwords per second), 66,351 kp/s의 처리량을 보였다. 동일한 환경에서 기존 최고 성능을 달성한 해시캣의 구현과 비교하여 각각 22.5, 15.2% 더 높은 처리량을 달성하였다.

2. 관련 연구

2.1 GPU상에서 암호 해독

최근 몇 년 동안 하드웨어 기술의 발전으로 많은 연구자가 GPU에서 암호 해독 알고리즘의 고속 구현에 관한 연구를 수행하였다. Geet al.[5]는 SHA 512 알고리즘의 철저한 공격을 위해 GPU 상에서 최적화를 진행하였다. Dürmuth와 Kranz[6]은 Bcrypt와 Scrypt 두 가지 유형의 해시 암호를 GPU와 FPGA에서 각각 구현했다. Chen et al.[7]은 MD5 Crypt 알고리즘의 대량 처리를 위해 CUDA 프로그래밍을 사용하여 최적화했다. CUDA 최적화 구현을 위해 잘 알려진 방법인 최적의 그리드당 블록의 수와 블록당 스레드 수 고려, 공유메모리 사용과 CUDA Streams 사용 기법을 사용하였다. Li et al.[8]는 CUDA를 사용하여 MD5-RC4을 효율적을 구현하였다. “S” 배열이 GPU 로컬 메모리에 배치됨을 분석하고 공유메모리를 사용하여 CPU 기반 구현보다 3-5배의 속도 향상을 보였다. Kim et al.[9]는 CUDA를 사용하여 PDF 암호 해독을 GPU 상에서 가속화 하였다. 특히 다중 GPU를 사용하여 CPU보다 1,000배 더 빠른 속도를 보였다. PDF 암호 해독은 대부분의 해독 도구에서 지원한다. 이중 해시캣은 많은 사용자를 보유하고 있으며 가장 좋은 성능으로 평가받는 암호 도구 중 하나이다. Jens “atom” Steube가 2009년에 처음 만든 해시캣 도구는 많은 사용자가 사용하는 오픈소스 프로젝트이다. 해시캣의 코드는 2015년에 MIT 라이선스에 따라 공개적으로 사용할 수 있게 되었다[10]. OpenCL 기반으로 거의 모든 OpenCL 호환된다. 따라서 CPU, GPU, FPGA, DSP 및 보조 프로세서에서 실행할 수 있다. 또한 300개 이상의 다양한 알고리즘을 지원한다. 본 논문에서 성능 비교를 위해 가장 인기 있으며 높은 성능을 보이는 해시캣을 함께 비교한다.

2.2 MD5

PDF 1.4 - 1.6 버전의 암호 해독에는 MD5[11]과 RC4[12]을 반복적으로 사용한다. MD5 는 임의 길이의 메시지를 입력받아 128비트의 값을 출력하는 암호화 해시 함수이다. 임의 길이의 입력 메시지는 512비트 블록들로 나뉘어 순서대로 연산 된다. 32-비트 워드 4개로 이루어진 128비트 State로 동작한다. 첫 State는 고정 상수로 초기화된다. 메시지 블록은 각 라운드에서 처리되며 라운드는 Fig. 1과 같이 비선형 함수 F, 모듈식 덧셈 및 왼쪽 회전을 기반으로 하는 연산으로 구성된다. M_i는 입력 메시지의 32비트 블록이며 K_i는 상수이다. 함수 F는 아래의 4가지로 나뉘며 64번의 라운드에서 16라운드마다 Equation (1)과 같이 다른 F 함수가 사용된다.

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee (\neg X \wedge Z) \\ G(X,Y,Z) &= (X \wedge Z)(Y \wedge \neg Z) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \vee \neg Z) \end{aligned} \tag{1}$$

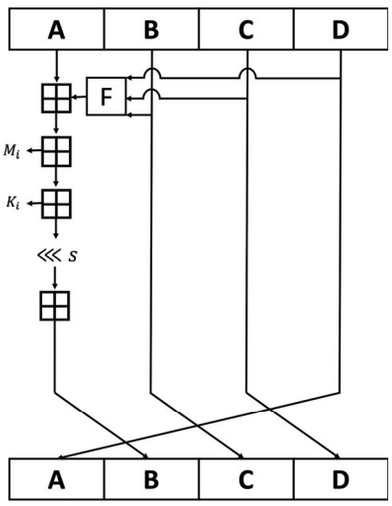

Fig. 1. MD5 Round Function

```
int KSA(char *key, unsigned char *S) {
    int len = strlen(key);
    int j = 0;

    for(int i = 0; i < N; i++)
        S[i] = i;

    for(int i = 0; i < N; i++) {
        j = (j + S[i] + key[i % len]) % N;

        swap(&S[i], &S[j]);
    }
    return 0;
}
```

Fig. 2. RC4 Key-Scheduling Algorithm Reference C Code

2.3 RC4

RC4는 키 스케줄링 알고리즘과 유사 난수 생성 알고리즘으로 구성된 스트림 암호로이다. 키 스케줄링 알고리즘은 256바이트의 순열과 2개의 8비트 인덱스 포인터를 사용하여 40~2048비트 사이의 가변 길이 키로 초기화된다. Fig. 2는 KSA의 C코드 구현이다. 이후 Fig. 3의 유사 난수 생성 알고리즘으로 비트 스트림이 생성된다.

```
int PRGA(unsigned char *S, char *plaintext, unsigned char *ciphertext)
    int i = 0;
    int j = 0;
    for(size_t n = 0, len = strlen(plaintext); n < len; n++) {
        i = (i + 1) % N;
        j = (j + S[i]) % N;

        swap(&S[i], &S[j]);
        int rnd = S[(S[i] + S[j]) % N];

        ciphertext[n] = rnd ^ plaintext[n];
    }
    return 0;
}
```

Fig. 3. RC4 Pseudo-Random Generation Algorithm Reference C Code

2.4 PDF 암호 해독 알고리즘

암호화된 파일에서 사용자가 인증을 위해 암호를 입력하면 암호화 해시를 계산하고 저장된 해시와 비교한다. 일치하는 경우 시스템은 사용자에게 지정된 자원에 대한 액세스 권한을 부여한다. 암호 크래킹의 경우 같은 확인 절차를 사용한다. 비밀번호 후보를 입력하고 출력을 얻으면 검증 값이라고도 하는 알려진 해시와 비교한다. 출력 값과 알려진 해시값이 일치하면 암호가 올바른 것으로 간주한다. 독점 응용 프로그램은 암호 처리의 내부 구현을 숨기는 경우가 많지만, 개방형 형식은 일반적으로 문서로 만들어져 있으며 필요한 암호 확인 단계의 사양은 대부분 공개적으로 사용할 수 있다. PDF에는 이러한 해시를 생성하는 알고리즘이 공개되어 있으며 PDF의 버전에 따라 다른 알고리즘이 사용된다[13]. 본 논문의 구현 대상인 PDF 1.4-1.6 버전의 암호 해독 알고리즘은 3단계로 나눌 수 있다. 첫 단계에서 추측 패스워드값은 패딩 값과 파일의 메타 데이터(objstring, permission, fileID)와 결합하여 MD5 알고리즘으로 해싱된다. 다음으로 128비트의 해싱 값은 MD5으로 50번 반복적으로 해싱된다. 다음 단계는 RC4 암호화 과정이 50회 반복된다. 반복적인 MD5 연산 과정으로 생성된 128비트 해싱 값과 상수값을 XOR하여 RC4 암호화의 키값으로 사용한다. 그리고 입력 평문으로 파일의 User_String 값을 사용하여 RC4으로 20회 암호화한다. 세 번째 과정에서는 fileID 값을 MD5로 해싱한 값과 이전 과정에서 생성된 암호문을 비교하여 패스워드 검증한다. 동일한 해시 값을 출력하는 패스워드가 나올 때까지 이전 단계를 반복한다. 알고리즘은 Fig. 4와 같다.

2.5 CUDA 프로그래밍

CUDA[13]는 C 프로그래밍 언어와 여러 산업 표준 언어를 사용하여 GPU에서 수행하는 병렬 처리 알고리즘 작성할 수 있도록 하는 병렬 컴퓨팅 플랫폼 및 응용 프로그래밍 인터페이스이다. CUDA GPU는 GPU에서 실행되는 함수 커널,

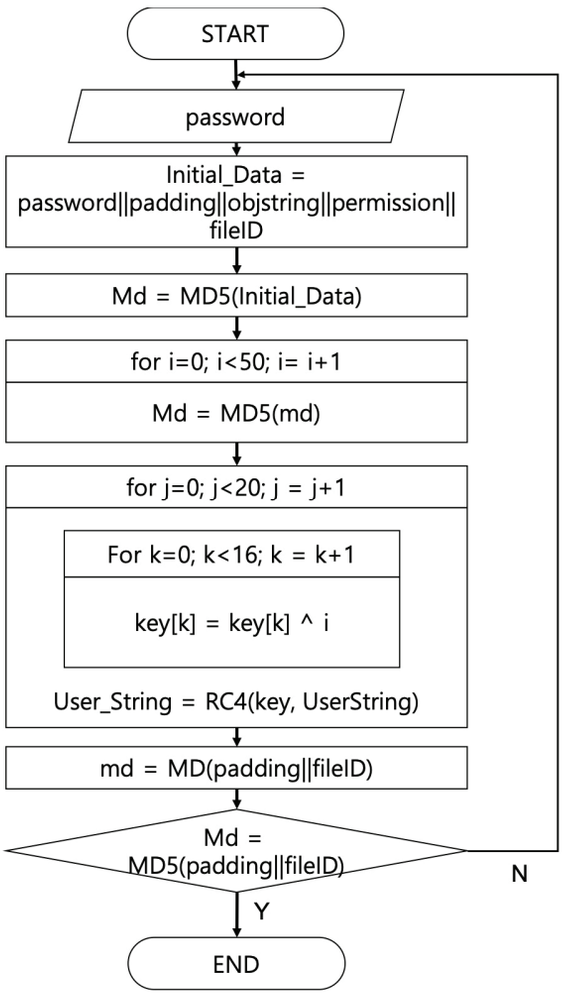

Fig. 4. Decryption Algorithms in PDF 1.4 -1.6 Versions

스레드 그룹 블록, 블록의 그룹 그리드, 32개의 스레드 묶음 워프, 1개의 워프가 실행되어 스레드를 동시에 실행하는 SM (Streaming Multi-processor)으로 구성된다. 성능을 위해 블록당 스레드 수, 그리드 당 블록 수의 적절한 값 선택 필요하다. CUDA의 큰 장점으로 스레드 간에 공유할 수 있는 빠른 공유메모리 영역에 접근할 수 있다. 본 논문의 제안기법에서는 공유메모리를 사용하여 높은 성능향상을 보일 수 있었다.

3. 제안기법

제안기법은 CUDA를 사용하여 PDF 1.4 -1.6 버전 암호 해독 알고리즘을 최적화한다. 암호 해독과정에 필요한 많은 양의 계산을 신속하게 처리하기 위해서는 많은 시간이 소모되는 부분에 대한 속도 최적화가 필요하다. 따라서 반복적으로 사용되는 MD5와 RC4를 중심적으로 최적화하였다.

[7]과 같이 단일적으로 알고리즘이 사용되며 입력이 가변적인 경우와 다르게 PDF 1.4 - 1.6 버전 암호 해독 알고리즘은 고정적인 128비트를 MD5로 반복적으로 해싱하고 이후 RC4는 반복된 MD5의 결과값 128비트를 입력받아 반복적으로 암호화한다. 제안기법은 고정적인 입력을 고려한 최적화를 제안한다. 이전연구 [8]에서는 RC4 알고리즘에서 상태 S를 공유메모리에 저장하는 기법을 사용한다. 그러나 공유메모리 크기의 제한으로 고정된 블록당 스레드 수를 사용한다. 본 논문에서는 높은 성능을 달성하는 블록당 스레드 수는 그리드당 블록 수에 따라 다른 것을 관찰하였다. 이에 대한 자세한 설명은 3.3에서 다룬다. [9]에서는 다중 GPU를 사용한 결과를 함께 제공한다. 그러나 암호 해독은 다수의 추측 비밀번호를 동일 알고리즘으로 처리하는 단순 병렬 문제이다. 따라서 다중 GPU의 최적화는 단일 GPU의 최적화와 동일하므로 본 논문에서는 다루지 않는다.

3.1 MD5 최적화

MD5의 state는 32-비트 워드 a, b, c, d로 이루어진 진다. state는 상수값으로 초기화 후 사용된다. 상수값과 연산되는 부분은 첫 라운드에서 사전 연산이 가능하다. 아래는 첫 라운드 연산 과정의 의사 코드이다.

```
a += K;
a  = a +x + f (b, c, d);
a  = rotl32 (a, s);
a += b;
```

x를 제외한 a, b, c, d, K, s의 값은 상수값이므로 아래와 같이 변경될 수 있다.

```
a = rotl32(x + 0xd76aa477, s) + 0xefcdab89;
```

다음은 MD5 반복 연산을 최적화하였다. 알고리즘에서 128비트 입력에 대한 해시 연산이 50회 반복적으로 수행된다. 따라서 512비트 입력 메시지 M (M_0,M_1,…,M_30, M_31)에서 128비트(M_0,M_1,M_2,M_3)만 변경된다. 128비트(M_0,M_1,M_2,M_3)을 제외한 나머지 값들은 변경되지 않아 일부 메시지 워드의 덧셈 연산을 생략할 수 있다. Fig. 5는 제안기법의 구현 코드이다.

3.2 RC4 최적화

RC4는 8비트 연산에 최적화되었기 때문에 일반적인 구현은 8비트 워드 연산으로 이루어진다. 적용한 첫 번째 RC4 최적화 기법은 RC4 알고리즘의 32비트의 워드 통합 구현이다. PDF 암호 해독 알고리즘에서 RC4의 키는 MD5 연산의 결과로 생성된 128비트를 사용한다. MD5 연산은 32비트 워드로 연산이 진행되어 GPU의 32비트 프로세서와 일치한다. 또한 MD5의 결과값인 128비트 키값은 4개의 32비트 워드 상태로 저장된다. 따라서 RC4의 연산을 32비트의 워드로 연산하면 8비트 워드로 변환 없이 RC4 이전 MD5 과정의 출력값

```
#define STEP(f,a,b,c,d,x,K,s)  \
{                              \
a += K;                        \
a  = a +x+ f (b, c, d);        \
a  = rotl32 (a, s);            \
a += b;                        \
}

//uint32_t a = 0x67452301;
uint32_t b = 0xefcdab89;
uint32_t c = 0x98badcfe;
uint32_t d = 0x10325476;

uint32_t a = rotl32(w[0] + 0xd76aa477, MD5S00)
+ 0xefcdab89;
STEP0(F, d, a, b, c, w[1] + MD5C01, MD5S01);
STEP0(F, c, d, a, b, w[2] + MD5C02, MD5S02);
STEP0(F, b, c, d, a, w[3] + MD5C03, MD5S03);
STEP0(F, a, b, c, d, 0x80 + MD5C04, MD5S00);
STEP0(F, d, a, b, c, MD5C05, MD5S01);
STEP0(F, c, d, a, b, MD5C06, MD5S02);
STEP0(F, b, c, d, a, MD5C07, MD5S03);
STEP0(F, a, b, c, d, MD5C08, MD5S00);
STEP0(F, d, a, b, c, MD5C09, MD5S01);
STEP0(F, c, d, a, b, MD5C0a, MD5S02);
STEP0(F, b, c, d, a, MD5C0b, MD5S03);
STEP0(F, a, b, c, d, MD5C0c, MD5S00);
STEP0(F, d, a, b, c, MD5C0d, MD5S01);
STEP0(F, c, d, a, b, 128 + MD5C0e, MD5S02);
STEP0(F, b, c, d, a, MD5C0f, MD5S03);
```

Fig. 5. Implementation of MD5's Proposed Technique

을 32비트 워드 그대로 사용할 수 있으며 결합한 만큼의 덧셈 및 XOR 연산 횟수를 줄일 수 있다. Fig. 6은 제안기법의 구현 코드이다.

RC4 알고리즘은 상태 배열 S를 KSA와 PRGA에서 반복적으로 조회한다. 이때 메모리에서 값을 저장하고 불러오는 과정에서 지연이 발생 한다. 이를 개선하기 위해 상태 배열 S를 공유메모리에 저장한다. 해당 방법으로 상태 배열 S 조회 및 저장에 생기는 지연을 최소화 할 수 있다. 하나의 상태 배열 S 256바이트가 필요하다. 각 스레드에서 하나의 암호 해독 알고리즘이 동작하므로 공유메모리 또한 각각의 스레드마다 상태 배열 S를 저장할 256바이트가 필요하다. 이러

```
out[0] = u_string[0];
out[1] = u_string[1];
out[2] = u_string[2];
out[3] = u_string[3];

uint32_t xv;
uint32_t tmp[4];

extern __shared__ uint32_t S[];

for (int k = 19; k >= 0; --k) {
    xv = k << 0 | k << 8 | k << 16 | k << 24;
    tmp[0] = digest[0] ^ xv;
    tmp[1] = digest[1] ^ xv;
    tmp[2] = digest[2] ^ xv;
    tmp[3] = digest[3] ^ xv;

    rc4_init_128(S, tmp);
    rc4_next_16(S, out, out);
}
```

Fig. 6. Implementation of RC4's Proposed Technique

한 점 때문에 커널에서 사용할 수 있는 최대 공유메모리 크기를 넘지 않도록 블록당 스레드의 수를 선택해야 한다. 구현에는 공유메모리를 동적 할당하여 구현하였다. 공유메모리는 정적 혹은 동적으로 할당할 수 있다. GPU 아키텍처마다 사용할 수 있는 공유메모리 용량이 다르므로 사용할 수 있는 스레드도 GPU 아키텍처마다 다르다. 이러한 점을 고려하여 제안기법에서는 Fig. 7과 같이 공유메모리를 동적 할당하여 구현하였다.

3.3 블록당 스레드, 그리드 당 스레드 수 조정

블록당 스레드 수와 그리드 당 블록 수는 성능에 영향을 미친다. 따라서 최적 성능을 달성하기 위해 적절한 선택이 필요하다. 제안기법에서 최적에 성능을 달성하는 블록당 스레드 수와 그리드 당 블록 수를 확인하기 위해 블록당 스레드 수와 그리드 당 블록 수에 따른 성능 변화를 확인하였다. RTX 3060 환경에서 제안기법을 적용한 PDF 암호 해독 구현을 실행하여 블록당 스레드 수와 그리드 당 블록 수 변화에 따른 초당 패스워드 연산 횟수를 측정하였다. Fig. 8은 블록 수에 따른 초당 패스워드 연산 횟수 결과이다. 블록 수가 증가할수록 블록 수에 따라 성능이 높아지다 특정 지점에 수렴하였다. 스레드 수에 따른 초당 패스워드 연산 횟수의 경우 블록의 수가 적을 때는 Fig. 9와 같이 일정하게 증가하였다. 그러나 블록의 수가 많아지면 Fig. 10과 같이 성능이 불규칙적이었다.

```
int maxbytes = deviceProp.sharedMemPerBlockOptin;
    cudaFuncSetAttribute(pdfcrack, cudaFuncAttributeMaxDynamicSharedMemorySize, maxbytes);

pdfcrack <<< gSize, tSize, 64 * tSize * sizeof(uint32_t) >>> (dev_passward, dev_passlen, dev_hash, dev_check, dev_o_string, dev_u_string, dev_pdata, dev_crack);
```

Fig. 7. Implementing Dynamic Allocation of Shared Memory

제안한 PDF 암호 해독 구현의 블록당 스레드 수와 그리드 당 블록 수에 따른 성능의 불규칙성을 고려하여 최적의 파라미터 선택을 위해 [14]의 Autotune을 사용하였다. Autotune은 성능에 영향을 미치는 레지스터 사용, 스레드 블록 크기, 루프 등과 같은 다양한 요인을 장치와 알고리즘에 따라 최적 파라미터 값을 자동으로 탐색하는 기법이다. [14] 은 레지스터 사용, 스레드 블록 크기, 루프 unroll과 같은 성능 요인을 탐색한다. 해시캣 또한 최적의 파라미터 선택을 위해 V3부터 Autotune 기능 추가하였다. 제안기법에서는 블록당 스레드 수와 그리드 당 블록 수 두 가지 요소를 탐색하였다. 제안 기법의 구현은 RTX 3060, RTX3090 환경에서 그리드 당 블록 수 65,536, 블록당 스레드 수 96에서 가장 좋은 성능을 보였다.

4. 성능 비교

본 장에서는 제안기법을 적용한 구현을 이전 연구인 [9]와 기존 최고 성능을 달성한 해시캣의 구현과 비교한다. Hashcat-6.2.5버전을 사용하였으며 해시캣에서 제공되는 벤치마크 모드(hashcat -b -m10500 -w 4 -O)로 동작한 초당 패스워드 계산 횟수를 RTX 3060, RTX 3090 환경에서 측정하였다. 해시캣은 GPU로 데이터 전송하기 이전부터 커널 완료까지 측정하여 초당 패스워드 계산 횟수를 측정한다. 따라서 제안기법의 구현도 같은 기준으로 측정하였다. 결과는 Talbe 1에서 확인 할 수 있다. 제안기법의 구현은 RTX 3060에서 31,460kp/s, RTX3090에서 66,351kp/s를 달성하였다. 이전 연구와는 각각 120배, 250배 이상의 처리량을 달성하였다. 그러나 이는 실험에 사용된 GPU의 성능 차이가 큰 이유로 판단된다. 같은 환경에서 동작시킨 해시캣과 비교했을 때 RTX 3060에서 22.5%, RTX 3090에서 15.2%의 높은 처리량을 달성하였다.

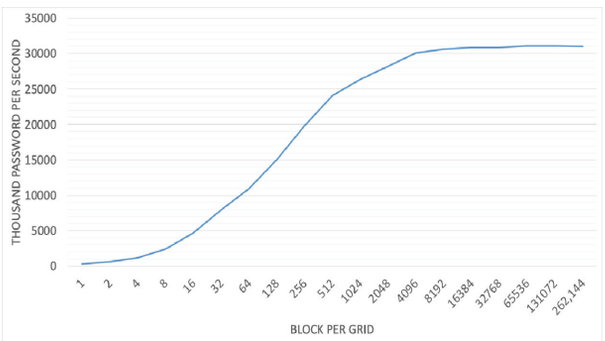

Fig. 8. When Cracking PDF in RTX 3060 Environment, the Number of Password Operations per Second According to the Number of Blocks per Grid(Kp/S)

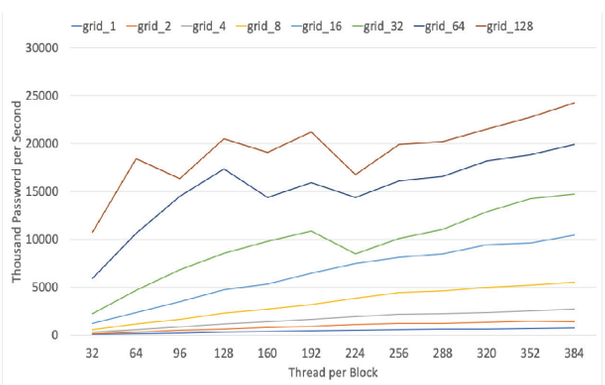

Fig. 9. When Cracking PDF in RTX 3060 Environment, the Number of Password Operations per Second (kp/s) According to the Number of Threads per Block at Each Block of 128 or Less per Grid

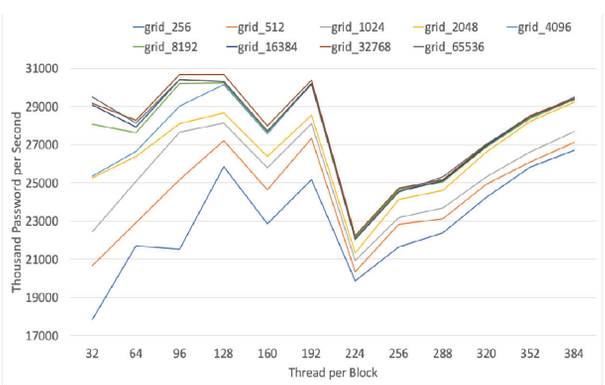

Fig. 10. When Cracking PDF in RTX 3060 Environment, the Number of Password Operations per Second (kp/s) According to the Number of Threads per Block in Each of 128 or More Blocks per Grid

Table 1. Comparison of Calculations per Second

Reference	Environment	Speed
[9]	9800GTX	262 kp/s
[13]	9800GTX	273 kp/s
[9]	4 * Tesla C1060	1,200 kp/s
hashcat 6.2.5	RTX 3060	25,693 kp/s
ours	RTX 3060	31,460 kp/s
hashcat 6.2.5	RTX 3090	57,601 kp/s
ours	RTX 3090	66,351 kp/s

5. 결 론

본 논문은 PDF 1.4 -1.6 버전 암호 해독의 알고리즘을 CUDA GPU 상에서 최적화 구현하였다. MD5, RC4의 최적을 위해 MD5 불필요 연산 제거, RC4의 32비트 워드 통합, 공유메모리 사용의 기법을 사용하였으며 Autotune을 사용하여 최적의 성능에 가까운 블록당 스레드 수, 그리드 당 블록 수의값을 탐색하여 높은 성능 향상을 보였다. 결과적으로 RTX 3060 환경에서 31,460kp/s의 처리량 RTX 3090 환경에서 66,351kp/s의 처리량을 달성하였다. 이는 이전 최고 성능인 크래킹 도구 해시캣 보다 각각 22.5%, 15.2% 향상된 처리량이다. 또한 성능향상의 영향을 미치는 블록, 스레드 수 조정을 위해 오토튠 기법을 사용하여 구현하였다. 추후 연구에는 다른 PDF 버전의 암호 해독 알고리즘의 최적화 구현과 다양한 환경에서 성능 비교를 하고자 한다.

References

[1] Du, Xiaoyu et al., "SoK: Exploring the state of the art and the future potential of artificial intelligence in digital forensic investigation," *Proceedings of the 15th International Conference on Availability, Reliability and Security*, No.46, pp.1-10, 2020.

[2] K. W. Kim and U. S. Kyong, "Efficient implementation of MD5 algorithm in password recovery of a PDF file," *2012 7th International Conference on Computing and Convergence Technology (ICCCT)*, IEEE, 2012.

[3] L. Bosnjak, J. Sres, and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (Mipro), pp.1161-1166, 2018.

[4] M. Hardy, L. Masinter, D. Markovic, D. Johnson, and M. Bailey, "The application/pdf Media Type," No.rfc8118, 2017.

[5] C. Ge, L. Xu, W. Qiu, Z. Huang, J. Guo, G. Liu, and Z. Gong, "Optimized password recovery for SHA-512 on GPUs," *2017 IEEE International Conference on Computational Science and Engineering(CSE) and IEEE International Conference on Embedded and Ubiquitous Computing(EUC)*, IEEE, Vol.2, pp.226-229, 2017.

[6] M. Dürmuth and T. Kranz, "On password guessing with GPUs and FPGAs," *International Conference on Passwords*. Springer, Cham, Vol.9393, pp.19-38, 2014.

[7] R. Chen, Y. Zhang, J. Zhang, and J. Xu, "Design and optimizations of the MD5 crypt cracking algorithm based on CUDA," *International Conference on Cloud Computing*, Springer, Cham, pp.155-164, 2014.

[8] C. Li, H. Wu, S. Chen, X. Li, and D. Guo, "Efficient implementation for MD5-RC4 encryption using GPU with CUDA," *2009 3rd International Conference on Anti-counterfeiting, Security and Identification in Communication*, IEEE, pp.167-170, 2009.

[9] K. W. Kim, S. S. Lee, D. W. Hong, and J. C. Ryou, "Gpu-accelerated password cracking of pdf files," *KSII Transactions on Internet and Information Systems(TIIS)*, Vol.5, No.11, pp.2235-2253, 2011.

[10] Hashcat, hashcat: World's fastest and most advanced password recovery utility, 2022, [Internet], Available: https://github.com/hashcat/hashcat.

[11] B. D. Boer and A. Bosselaers. "Collisions for the compression function of MD5, Advances in Cryptology," *Workshop on the Theory and Application of of Cryptographic Techniques*, pp.293-304, 1993.

[12] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege, "Analysis methods for (alleged) RC4," *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Berlin, Heidelberg, pp.327-341, 1998.

[13] S. Cook, "CUDA programming: A developer's guide to parallel computing with GPUs," Newnes, 2012.

[14] A. Khalid, G. Paul, and A. Chattopadhyay, "New speed records for Salsa20 stream cipher using an autotuning framework on GPUs," *International Conference on Cryptology in Africa*, Springer, Berlin, Heidelberg, pp.189-207, 2013.

김 현 준

https://orcid.org/0000-0002-6847-6772
e-mail : khj930704@gmail.com
2019년 한성대학교 IT응용시스템공학부 (학사)
2021년 한성대학교 IT융합공학부(석사)
2021년 ~ 현 재 한성대학교 정보컴퓨터공학과 박사과정
관심분야 : 정보보호, 암호화 구현, 부채널분석

엄 시 우

https://orcid.org/0000-0002-9583-5427
e-mail : shuraatum@gmail.com
2021년 한성대학교 IT융합공학부(학사)
2021년 ~ 현 재 한성대학교 IT융합공학부 석사과정
관심분야 : 정보보호, 암호 구현, 인공지능

서 화 정

https://orcid.org/0000-0003-0069-9061
e-mail : hwajeong84@gmail.com
2010년 부산대학교 컴퓨터공학과(학사)
2012년 부산대학교 컴퓨터공학과(석사)
2016년 부산대학교 컴퓨터공학과(박사)
2016년 ~ 2017년 싱가포르 과학기술청 연구원
2019년 ~ 현 재 한성대학교 IT융합공학부 조교수
관심분야 : 정보보호, 암호구현