GPU 상에서 경량암호 SIMON의 비트슬라이싱 구현

김현준*, 서화정**

*한성대학교 (대학원생)

**한성대학교 (교수)

Bit slicing implementation of lightweight cipher SIMON on GPU

Hyun-jun Kim*, Hwa-seong Seo**

*Information and Computer Engineering(Graduate student)

**Department of IT Convergence Engineering(professor)

요 약

IoT와 클라우드 컴퓨팅 기술의 도래로 수많은 데이터가 생성되고 전송되고 있다. 이러한 통신에 필요한 고속 암호화의 필요성이 대두됨에 따라 GPU의 높은 처리 능력을 기반으로 한 암호화 처리에 그래픽 프로세서를 활용하는 연구가 진행되고 있다. 본 논문에서는 경량 블록암호 SIMON에 비트슬라이싱 기법을 적용하여 GPU 상에서 구현하였다. SIMON 64/128을 대상으로 구현하였으며 결과적으로 RTX 3060상에서 38.651Gbps의 높은 처리량을 달성하였다.

I. 서론

최근 몇 년 동안 IoT 기술의 눈부신 성장으로 일상생활에 많은 스마트 애플리케이션이 도입되었다. 이 때문에 IoT 기기 간에 교환되는 데이터가 전례 없는 규모로 증가하고 있으며, 대량 데이터 통신에서 데이터 보안의 필요성이 주목받고 있다. 이러한 대량 데이터 통신에 필요한 고속 암호화의 필요성이 대두됨에 따라 GPU의 높은 처리 능력을 기반으로 한 암호화 처리에 그래픽 프로세서를 활용하는 연구가 진행되고 있다. GPU를 사용함으로 암복호화를 높은 처리량을 달성할 수 있으며 다른 작업을 위해 CPU를 확보할 수 있다. 본 논문에서는 경량 블록암호 SIMON에 비트슬라이싱 기법을 적용하여 GPU 상에서 구현하였다. SIMON 64/128을 대상으로 구현하였으며 결과적으로 RTX 3060상에서 38.651Gbps의 높은 처리량을 달성하여 이는 기존 Jan et. al[3]의 RTX 2060에서 2.064Gbps보다 높은 성능을 달성하였다.

II. 경량 블록암호 SIMON

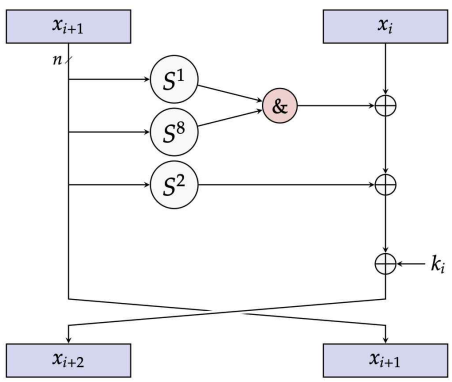

그림 1. SIMON의 라운드 함수[1]

SIMON[1]은 NSA에서 고도로 제한된 장치의 보안 문제를 해결과 하드웨어의 고성능을 위해 설계한 Feistel 구조의 경량 블록 암호 제품군이다. SIMON 블록 암호는 다양한 블록 크기와 키 길이를 지원하며 Simon2n/nm으로 표시된

III.

다. Simon64/128은 128비트 키를 사용하는 64 비트 평문으로 동작한다.

SIMON의 라운드 함수는 그림 1과 같이 동작한다. ⊕는 비트 논리연산 XOR, &은 AND , Sj는 j비트 왼쪽 회전 연산이다.

SIMON의 키 스케줄은 그림 2의 수식과 같다.

$$k_{i+m} = \begin{cases} c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}k_{i+1}), & m = 2 \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}k_{i+2}), & m = 3 \\ c \oplus (z_j)_i \oplus k_i \oplus (I \oplus S^{-1})(S^{-3}k_{i+3} \oplus k_{i+1}), & m = 4 \end{cases}$$

그림 2. SIMON의 키 스케줄 수식

IV. 제안기법

제안기법에는 SIMON에 비트 슬라이싱 기법을 적용하여 GPU 상에서 구현한다. 비트 슬라이싱은 DES의 소프트웨어 구현 속도를 높이기 위해 조회 테이블 대신 Biham[2]에서 처음 사용하였다. 비트 슬라이싱은 각 비트를 여러 블록으로 모아서 비트 단위로 연산하는 기법으로 여러 블록을 병렬로 처리할 수 있다.

비트슬라이싱 적용을 위해서는 연산 과정을 AND, OR, NOT과 같은 단순한 논리 게이트의 조합으로 변환해야 한다. 또한 데이터를 다중 블록 비트 슬라이스 표현으로 변환하는 패킹 과정과 원래의 표현으로 돌아가기 위한 언패킹 과정이 필요하다. GPU 코어가 32비트 아키텍처에 맞추어 SIMON 64/128을 32개의 64비트 일반 텍스트 블록이 병렬로 구현하였다.

2.1 라운드 키 update

32개의 블록이 병렬로 연산 되므로 많은 수의 라운드 키가 필요하다. 라운드 키를 메모리에 저장하게 되면 메모리에서 레지스터로 불러오는 추가적인 지연이 발생한다. 제안 기법에서는 on-the-fly 방식으로 구현하여 암호 동작과 함께 라운드 키를 연산한다. 라운드 키는 4라운드 동작마다 그림 3의 simon_update 함수를 통해 변경된다. 비트슬라이싱 기법을 적용하였으므로 상수값은 패킹 되어 비트슬라이싱 표현으로 입력받는다. 또한 비트 간의 연산이 진행되므로 기존의 비트 회전 연산은 해당 비트가 저장된 변수를 연산하는 것으로 대체되어 생략된다.

```
__constant__ uint32_t z[64] = { 0xffffffff, 0xffffffff, 0x0, 0xffffffff,
0xffffffff, 0x0, 0xffffffff, 0xffffffff, 0xffffffff, 0x0, 0xffffffff, 0x0,
0xffffffff, 0xffffffff, 0x0, 0x0, 0x0, 0xffffffff, 0xffffffff, 0x0, 0x0,
0xffffffff, 0x0, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0xffffffff, 0x0, 0x0, 0xffffffff, 0x0, 0x0, 0x0,
0xffffffff, 0x0, 0xffffffff, 0x0, 0x0, 0xffffffff, 0xffffffff, 0xffffffff,
0x0, 0x0, 0xffffffff, 0xffffffff, 0x0, 0xffffffff, 0x0, 0x0, 0x0, 0x0,
0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff};
__device__ void simon_update(uint32_t* K, uint32_t i) {
//RK0
K[0] = K[0] ^ K[(3) % 32 + 96] ^ K[32] ^ K[(4) % 32 + 96] ^ K[(1) % 32
+ 32];
K[1] = K[1] ^ K[(1 + 3) % 32 + 96] ^ K[1 + 32] ^ K[(1 + 4) % 32 + 96]
^ K[(1 + 1) % 32 + 32];
for (int j = 2; j < 32; j++)
K[j] = K[j] ^ K[(j + 3) % 32 + 96] ^ K[j + 32] ^ K[(j + 4) % 32 + 96] ^
K[(j + 1) % 32 + 32] ^ 0xfffffffffffc;
K[0] ^= z[i];
K[0 + 32] = K[0 + 32] ^ K[(0 + 3) % 32] ^ K[0 + 64] ^ K[(0 + 4) % 32]
^ K[(0 + 1) % 32 + 64];
K[1 + 32] = K[1 + 32] ^ K[(1 + 3) % 32] ^ K[1 + 64] ^ K[(1 + 4) % 32]
^ K[(1 + 1) % 32 + 64];
//RK1
for (int j = 2; j < 32; j++) {
K[j + 32] = K[j + 32] ^ K[(j + 3) % 32] ^ K[j + 64] ^ K[(j + 4) % 32] ^
K[(j + 1) % 32 + 64] ^ 0xfffffffffffc;
}
K[32] ^= z[i + 1];
//RK2
K[0 + 64] = K[0 + 64] ^ K[(0 + 3) % 32 + 32] ^ K[0 + 96] ^ K[(0 + 4)
% 32 + 32] ^ K[(0 + 1) % 32 + 96];
K[1 + 64] = K[1 + 64] ^ K[(1 + 3) % 32 + 32] ^ K[1 + 96] ^ K[(1 + 4)
% 32 + 32] ^ K[(1 + 1) % 32 + 96];
for (int j = 2; j < 32; j++) {
K[j + 64] = K[j + 64] ^ K[(j + 3) % 32 + 32] ^ K[j + 96] ^ K[(j + 4) %
32 + 32] ^ K[(j + 1) % 32 + 96] ^ 0xfffffffffffc;
}
K[64] ^= z[i + 2];
//RK3
K[0 + 96] = K[0 + 96] ^ K[(0 + 3) % 32 + 64] ^ K[0] ^ K[(0 + 4) % 32 +
64] ^ K[(0 + 1) % 32];
K[1 + 96] = K[1 + 96] ^ K[(1 + 3) % 32 + 64] ^ K[1] ^ K[(1 + 4) % 32 +
64] ^ K[(1 + 1) % 32];
for (int j = 2; j < 32; j++) {
K[j + 96] = K[j + 96] ^ K[(j + 3) % 32 + 64] ^ K[j] ^ K[(j + 4) % 32 +
64] ^ K[(j + 1) % 32 ] ^ 0xfffffffffffc;
}
K[96] ^= z[i + 3];
}
```

그림 3. 제안기법의 SIMON 라운드키 업데이트 구현

2.2 라운드 함수 F

SIMON의 F 함수는 비트 논리 연산으로 동작하므로 동일하게 사용된다. 그림 4.와 같이 구현하였으며 각 32개의 비트가 연산 되며 기존의 비트 회전 연산은 회전 후 연산될 비트를 사용하는 것으로 대체되어 생략된다.

```
__device__ void simon_f32(uint32_t* X, uint32_t* Y ) {
  Y[0]  ^= (X[31] & X[24]) ^ X[30];  Y[1]  ^= (X[0] & X[25]) ^ X[31];
  Y[2]  ^= (X[1] & X[26]) ^ X[0];    Y[3]  ^= (X[2] & X[27]) ^ X[1];
  Y[4]  ^= (X[3] & X[28]) ^ X[2];    Y[5]  ^= (X[4] & X[29]) ^ X[3];
  Y[6]  ^= (X[5] & X[30]) ^ X[4];    Y[7]  ^= (X[6] & X[31]) ^ X[5];
  Y[8]  ^= (X[7] & X[0]) ^ X[6];     Y[9]  ^= (X[8] & X[1]) ^ X[7];
  Y[10] ^= (X[9] & X[2]) ^ X[8];     Y[11] ^= (X[10] & X[3]) ^ X[9];
  Y[12] ^= (X[11] & X[4]) ^ X[10];   Y[13] ^= (X[12] & X[5]) ^ X[11];
  Y[14] ^= (X[13] & X[6]) ^ X[12];   Y[15] ^= (X[14] & X[7]) ^ X[13];
  Y[16] ^= (X[15] & X[8]) ^ X[14];   Y[17] ^= (X[16] & X[9]) ^ X[15];
  Y[18] ^= (X[17] & X[10]) ^ X[16];  Y[19] ^= (X[18] & X[11]) ^ X[17];
  Y[20] ^= (X[19] & X[12]) ^ X[18];  Y[21] ^= (X[20] & X[13]) ^ X[19];
  Y[22] ^= (X[21] & X[14]) ^ X[20];  Y[23] ^= (X[22] & X[15]) ^ X[21];
  Y[24] ^= (X[23] & X[16]) ^ X[22];  Y[25] ^= (X[24] & X[17]) ^ X[23];
  Y[26] ^= (X[25] & X[18]) ^ X[24];  Y[27] ^= (X[26] & X[19]) ^ X[25];
  Y[28] ^= (X[27] & X[20]) ^ X[26];  Y[29] ^= (X[28] & X[21]) ^ X[27];
  Y[30] ^= (X[29] & X[22]) ^ X[28];  Y[31] ^= (X[30] & X[23]) ^ X[29];
}
```

그림 4. 제안기법의 SIMON F함수 구현

2.3 Add RoundKey

SIMON의 Add RoundKey는 XOR 연산이므로 비트슬라이싱 기법에서도 동일하게 XOR 연산이 이루어진다. 그림 5와 같이 블록의 32비트에 라운드키 값이 XOR 되어 더 해진다.

```
__device__ void simon_addRoundKey(uint32_t* X, uint32_t* K) {
  X[0] ^= K[0], X[1] ^= K[1], X[2] ^= K[2], X[3] ^= K[3];
  X[4] ^= K[4], X[5] ^= K[5], X[6] ^= K[6], X[7] ^= K[7];
  X[8] ^= K[8], X[9] ^= K[9], X[10] ^= K[10], X[11] ^= K[11];
  X[12] ^= K[12], X[13] ^= K[13], X[14] ^= K[14], X[15] ^= K[15];
  X[16] ^= K[16], X[17] ^= K[17], X[18] ^= K[18], X[19] ^= K[19];
  X[20] ^= K[20], X[21] ^= K[21], X[22] ^= K[22], X[23] ^= K[23];
  X[24] ^= K[24], X[25] ^= K[25], X[26] ^= K[26], X[27] ^= K[27];
  X[28] ^= K[28], X[29] ^= K[29], X[30] ^= K[30], X[31] ^= K[31];
}
```

그림 5. 제안기법의 SIMON Add RoundKey 구현

V. 성능평가

제안기법의 구현 환경은 GPU는 RTX 3060를 사용하였으며 툴은 Visual Studio2019에서 CUDA를 사용하였다. 높은 처리량을 달성하는 그리드 당 블록 수와 블록당 스레드 수를 탐색하였다. 결과적으로 블록당 스레드 수 65536, 그리드 당 블록 수 256에서 38.651Gbps의 처리량을 달성하였다. 표 1.과 같이 이전연구인 Jang et. al[3]의 RTX2060에서의 2.064Gbps보다 약 18배 높은 성능이다.

	Giga bits per second(Gbps)	improvement
[3]	2.064	0%
our	38.651	1772%

표 1. 기존 대비 성능향상 비교

VI. 결론

본 논문에서는 경량 블록암호 SIMON에 비트슬라이싱 기법을 적용하여 GPU 상에서 구현하였다. SIMON 64/128을 대상으로 구현하였으며 결과적으로 RTX 3060상에서 38.651Gbps의 처리량을 달성하였다.

VII. Acknowledgment

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, 100%).

[참고문헌]

[1] Wetzels, Jos, and Wouter Bokslag. "Simple SIMON: FPGA implementations of the SIMON 64/128 Block Cipher." arXiv preprint arXiv:1507.06368 (2015).

[2] E. Biham, “A fast new des implementation in software,” in International Workshop on Fast Software Encryption, pp. 260 - 272, Springer,

[3] 장경배, 김현준, 임세진 and 서화정. "NVIDIA CUDA PTX를 활용한 SPECK, SIMON, SIMECK 병렬 구현" 정보보호학회논문지 31, no.3 (2021) : 423-431.doi: https://doi.org/10.13089/JKIISC.2021.31.3.423