ARX-based Korean Block Ciphers with CTR Mode and CTR_DRBG on Embedded Processors and GPU

국 가 암 호 공 모 전

CONTENTS

국 가 암 호 공 모 전

Introduction

ARX-based Korean Block Ciphers with CTR Mode and CTR_DRBG on Embedded Processors and GPU

국 가 암 호 공 모 전

Introduction: Target block cipher

- 본 논문에서는 국산 블록암호의 원활한 보급과 그 우수성을 설파하기 위해 국산 블록암호를 선택
  - 각각의 블록암호에 대한 기본 정보는 [표 1]을 따름

CHAM	HIGHT	LEA
2017년 발표 2019년 개정판 발표	2005년 발표 2006년 TTA 표준 제정 2010년 ISO/IEC 표준 제정	2013년 발표 2013년 TTA 표준 제정 2015년 암호모듈 검증제도 대상 알고리즘
CHAM-64/128 CHAM-128/128 CHAM-128/256	HIGHT-64/128	LEA-128/128 LEA-128/192 LEA-128/256

Table 1. Target block ciphers summarization

- 대상 블록암호를 8-bit AVR 마이크로컨트롤러와 GPU 상에서 최적 구현

Introduction: CTR mode of operation

- 블록암호 운용모드 중, 카운터 모드를 적용하여 최적 구현을 시도
  - 평문 대신 고정 값인 논스(Nonce)와 변수인 카운터(Counter) 값을 입력으로 사용하는 모드
  - 암호화와 복호화 구조가 동일하며 병렬처리가 가능한 운용모드

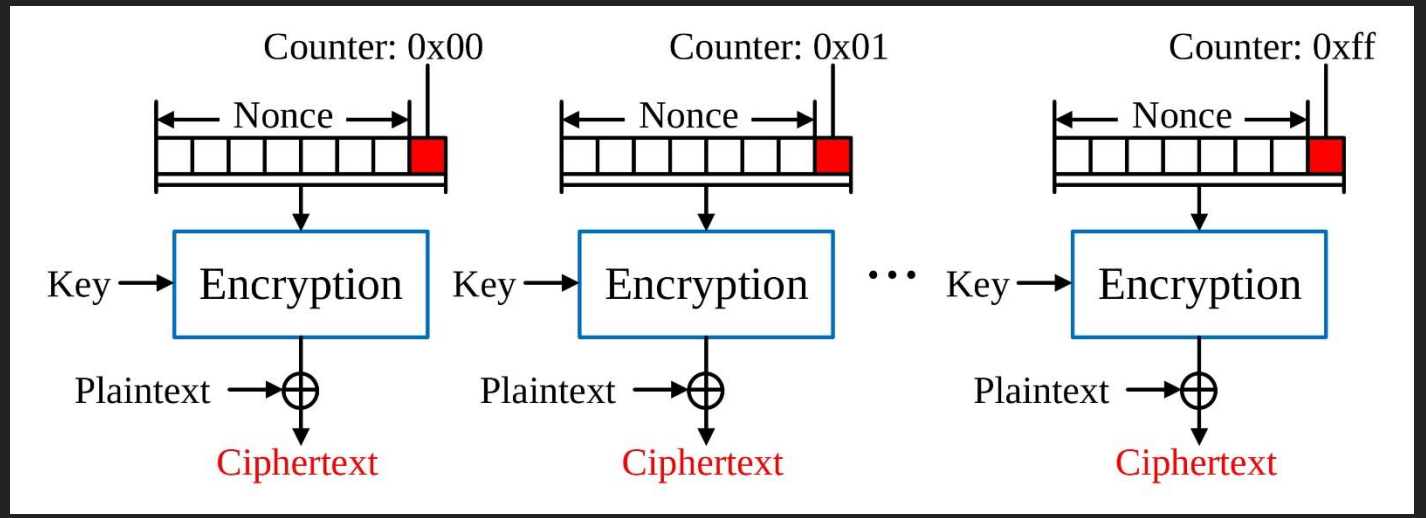

Fig 1. Encryption structure for counter mode of operation

Introduction: CTR mode of operation

- 카운터 모드의 논스 부분은 고정이라는 특성을 활용
  - 논스를 사용한 연산 과정에서, 다른 값들도 모두 고정일 경우 연산 결과는 항시 동일
  - 연산 결과가 동일하다면 결과 값을 미리 계산한 후 불러오는 것으로 일부 연산자를 생략
  - 이를 사전연산 기법으로 칭함

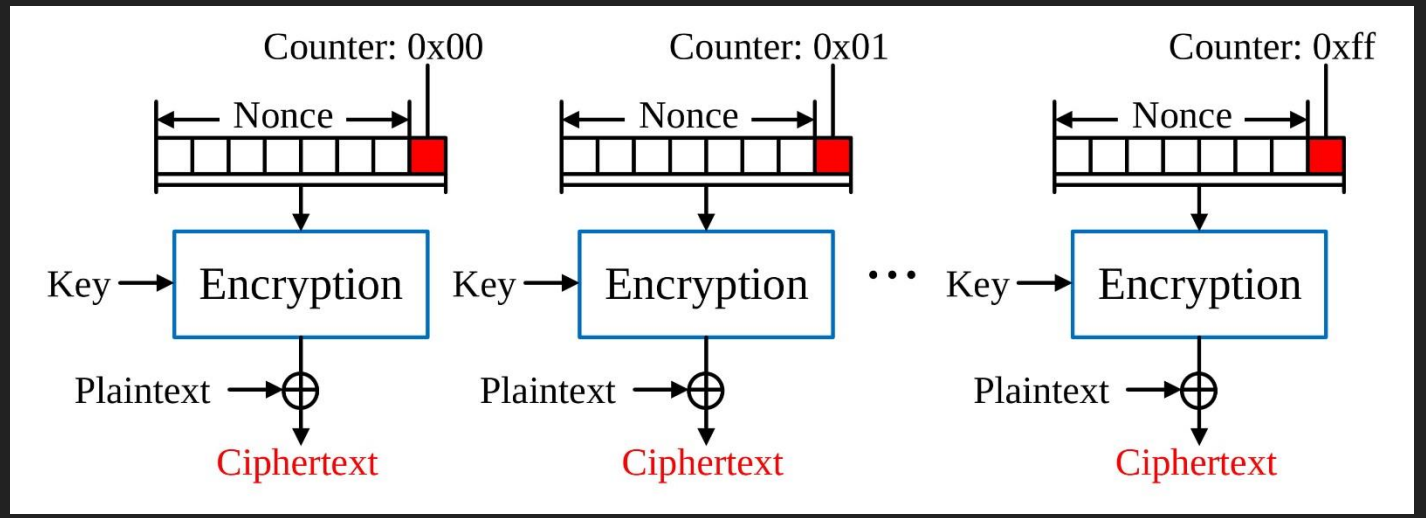

Fig 1. Encryption structure for counter mode of operation

Introduction: CTR mode of operation

- GPU 환경에서 카운터 모드의 카운터 부분은 블록마다 1씩 증가한다는 특성을 활용
  - 평문 대신 GPU의 각 스레드가 보유한 고유 번호(Thread ID)를 카운터 값으로 사용 가능
  - 사전에 평문을 CPU에서 GPU로 복사해야 하는 메모리 복사 시간을 생략

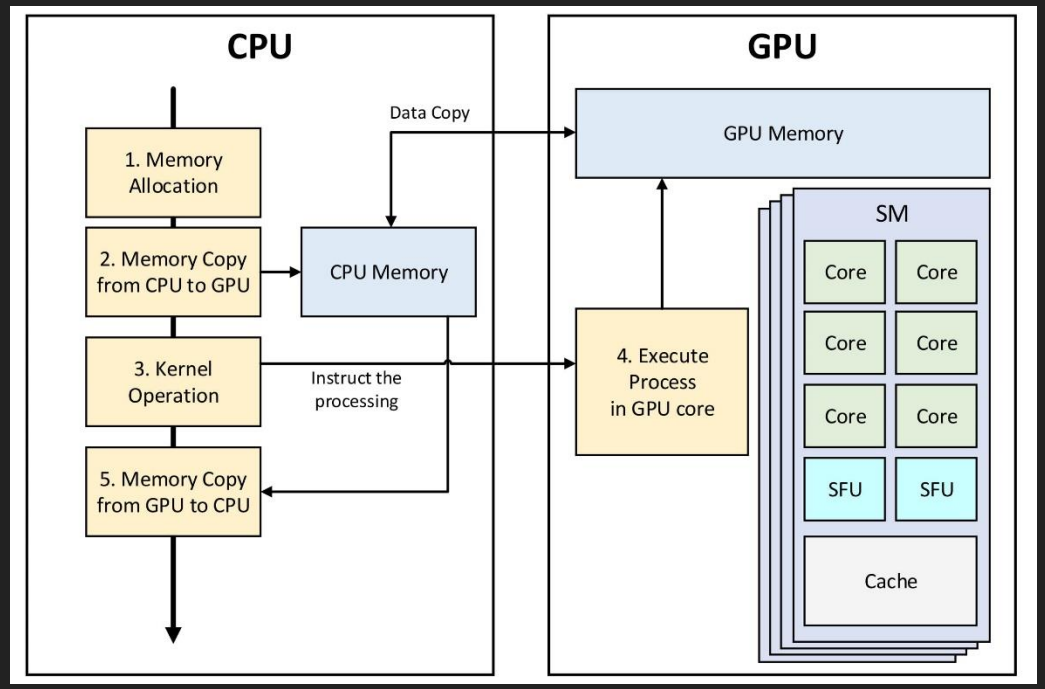

Fig 2. Memory copying process between CPU and GPU

Introduction: CTR_DRBG

- 블록암호 최적화 방안을 효과적으로 응용할 수 있는 CTR_DRBG를 선택
  - 인스턴스 생성함수 (Instantiate Function)를 통해 내부상태 (Internal State)를 갱신함
  - 블록암호에 따른 내부상태는 [표 2]를 따름
  - 출력생성 함수(Generate Function)는 내부상태를 이용해 난수 출력

Internal State	CHAM	LEA	HIGHT
C (Key Bit)	128/256	128/ 192/256	128
V (Block Bit)	64/128	128	64

Table 2. Parameters of Internal State for Block Cipher

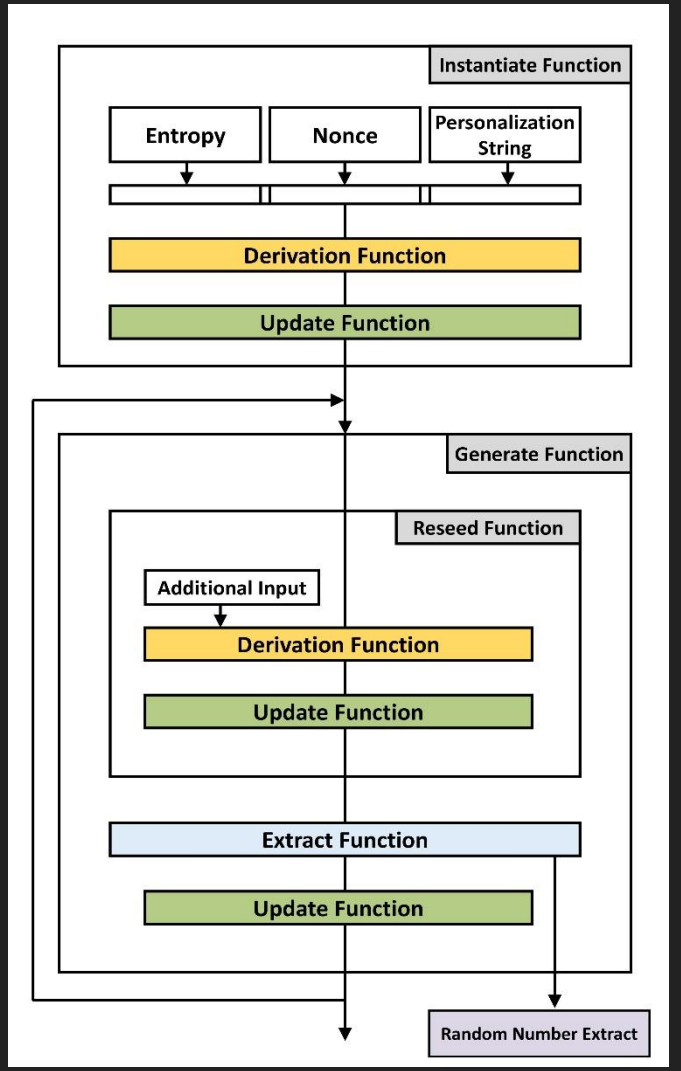

Fig 3. Overview of CTR_DRBG

Introduction: CTR_DRBG

- CTR_DRBG의 주요함수로는 유도함수(Derivation Function), 내부 갱신함수(Update Function), 출력함수(Extract Function)가 존재함
- 유도함수와 내부 갱신함수에서 CTR_DRBG 초기 내부상태가 상수임을 이용하여 사전연산이 가능함
- 출력함수에서 사용되는 CTR mode 암호화에 블록암호 최적화 방법, GPU 최적화 방법 적용가능

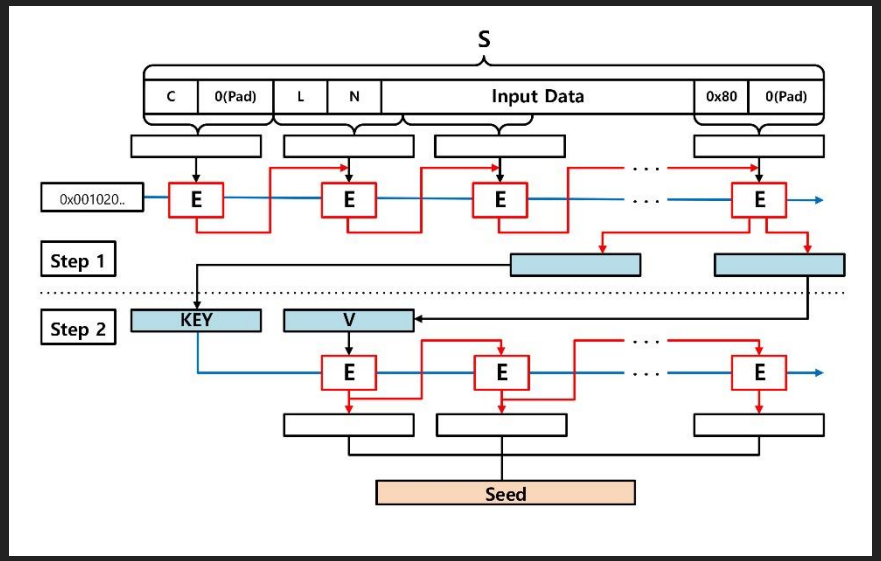

Fig 4. Derivation Function Structure

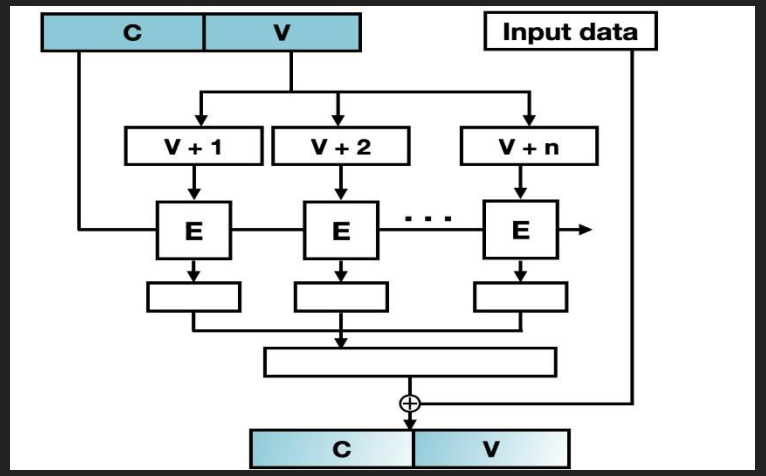

Fig 5. Update Function Structure

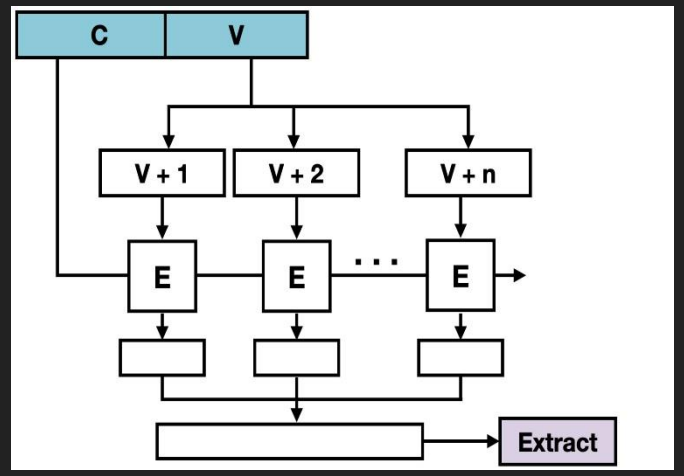

Fig 6. Extract Function Structure

Introduction: CTR_DRBG

- CTR_DRBG의 출력 함수에서 카운터 모드를 사용하는 특성을 활용
  - GPU에서 병렬연산을 활용하여 병렬적으로 난수 출력
  - 이를 CTR_DRBG GPU 최적화 기법으로 칭함

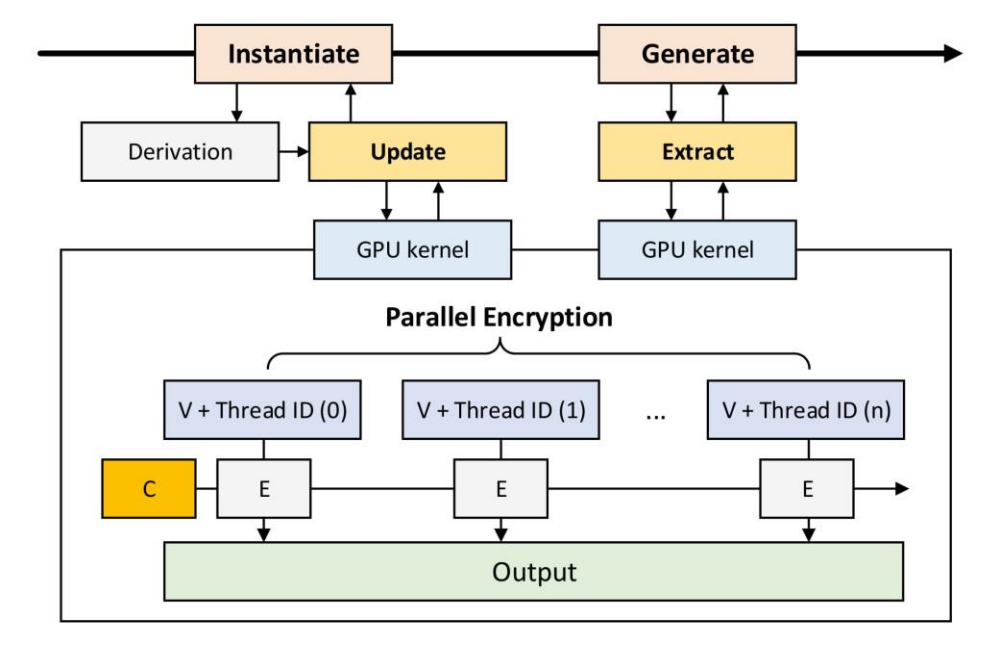

Fig 7. Parallel encryption structure for extract function

Our Works

ARX-based Korean Block Ciphers with CTR Mode and CTR_DRBG on Embedded Processors and GPU

국 가 암 호 공 모 전

Our Works: CTR mode optimization

- 카운터 모드의 특성을 사용하여 사전연산 기법을 적용
- 논스를 사용한 지점을 사전연산하여 사전연산 테이블(Look Up Table) 형태로 저장
- 라운드 함수 동작 시, 사전연산한 부분은 LUT에서 값을 호출
  - 사전연산 값 호출 부분에서는 라운드 함수 일부 연산자 생략 가능
- 키의 변경 유무에 따라 두 가지의 시나리오를 고려
  - 고정키 시나리오(Fixed-key)
  - 가변키 시나리오(Variable-key)
- 각각의 암호마다 내부 구조가 다르므로 다음 사양은 암호 알고리즘에 의존적
  - 사전연산 적용 최대 라운드
  - LUT의 크기

Our Works: CTR mode optimization

Incase of: CHAM block cipher

- CHAM은 3가지 규격이 있으며 내부 파라미터는 [표 3]을 따름
- 0~7라운드까지 사전연산이 가능

Cipher	n (block size)	k (key size)	rk (round key size)	r (number of rounds)
CHAM-64/128	64-bit	128-bit	16-bit	88
CHAM-128/128	128-bit	128-bit	32-bit	112
CHAM-128/256	128-bit	256-bit	32-bit	120

Table 3. Parameters of CHAM block cipher

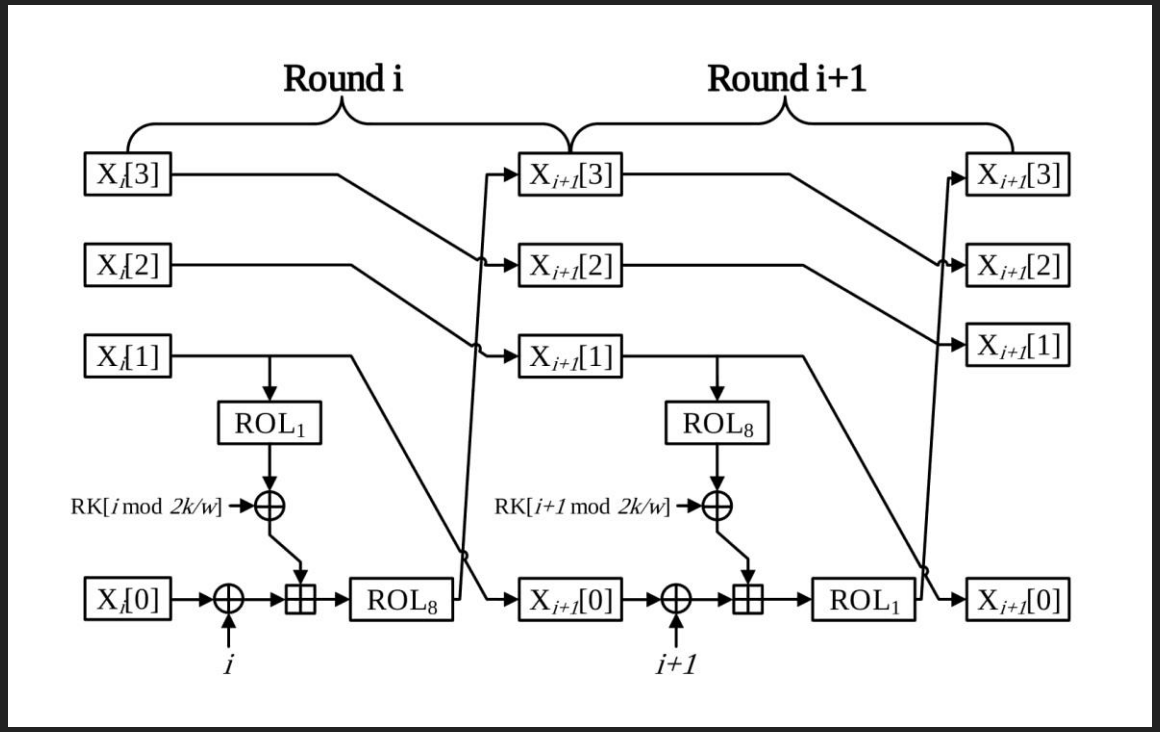

Fig 8. Encryption round function structure of CHAM block cipher

Our Works: CTR mode optimization

Incase of: CHAM block cipher

- CHAM에서 다음 값들을 사용하는 경우 사전연산 가능
  - 논스
  - 라운드 키
  - 라운드 카운터
- [그림 9]의 붉은 부분은 카운터 값의 흐름
  - 카운터 부분은 사전연산 불가능
  - 다른 암호 알고리즘도 이를 기본으로 따름
- 8라운드도 사전연산이 가능하나 제외됨
  - 너무 낮은 성능 향상
  - 8-way 기반 구현이 깨짐

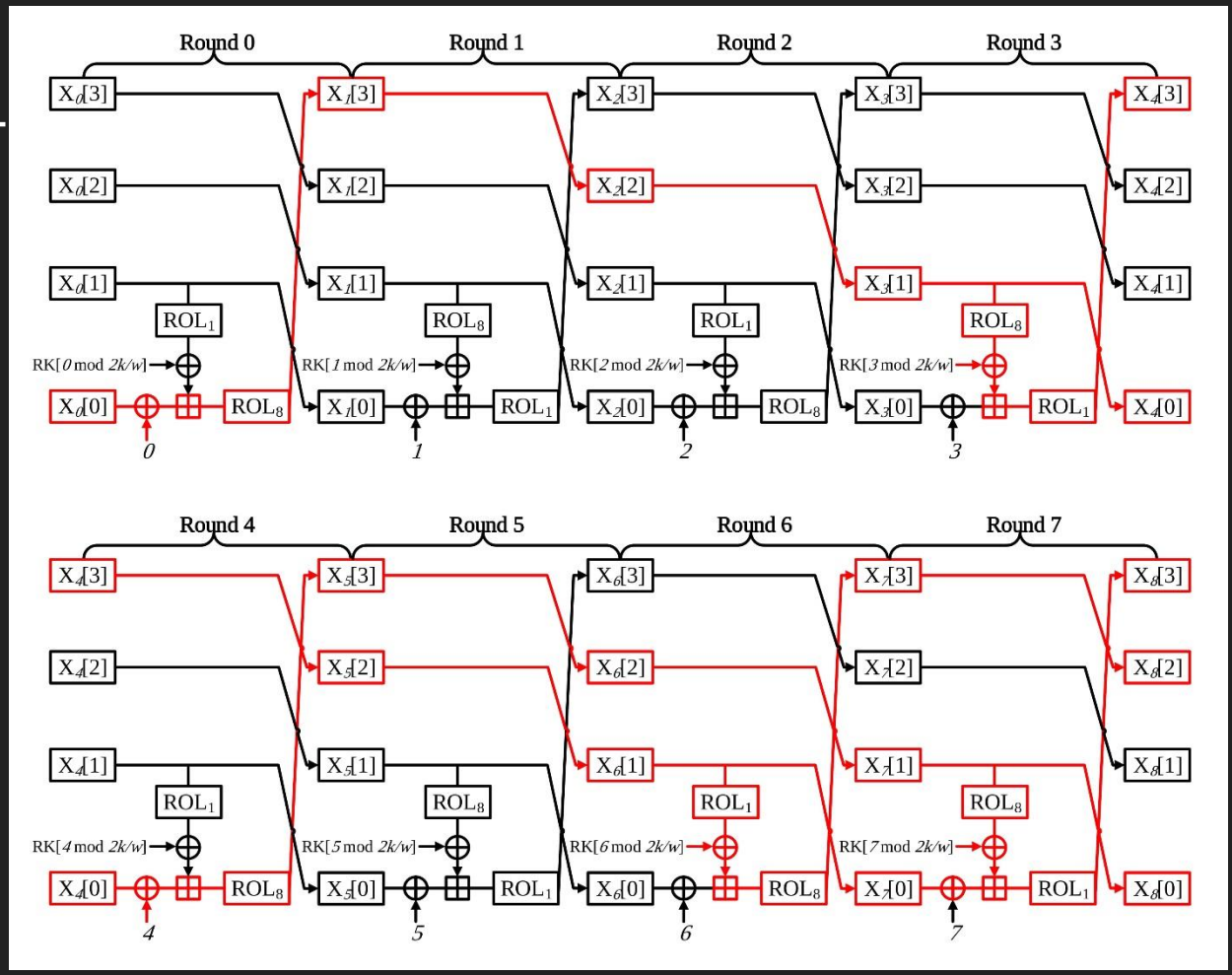

Fig 9. Counter value flow of CHAM block cipher

Our Works: CTR mode optimization

Incase of: CHAM block cipher

- [그림 10] 형태로 라운드 함수 재구성
- 생략된 명령어
  - LSL: 5개
  - ROL: 5개
  - ADD: 3개
  - ADC: 8개
  - XOR: 15개
  - ANDI: 1개
  - MOVW: 5개
  - LD: 10개
  - INC: 6개

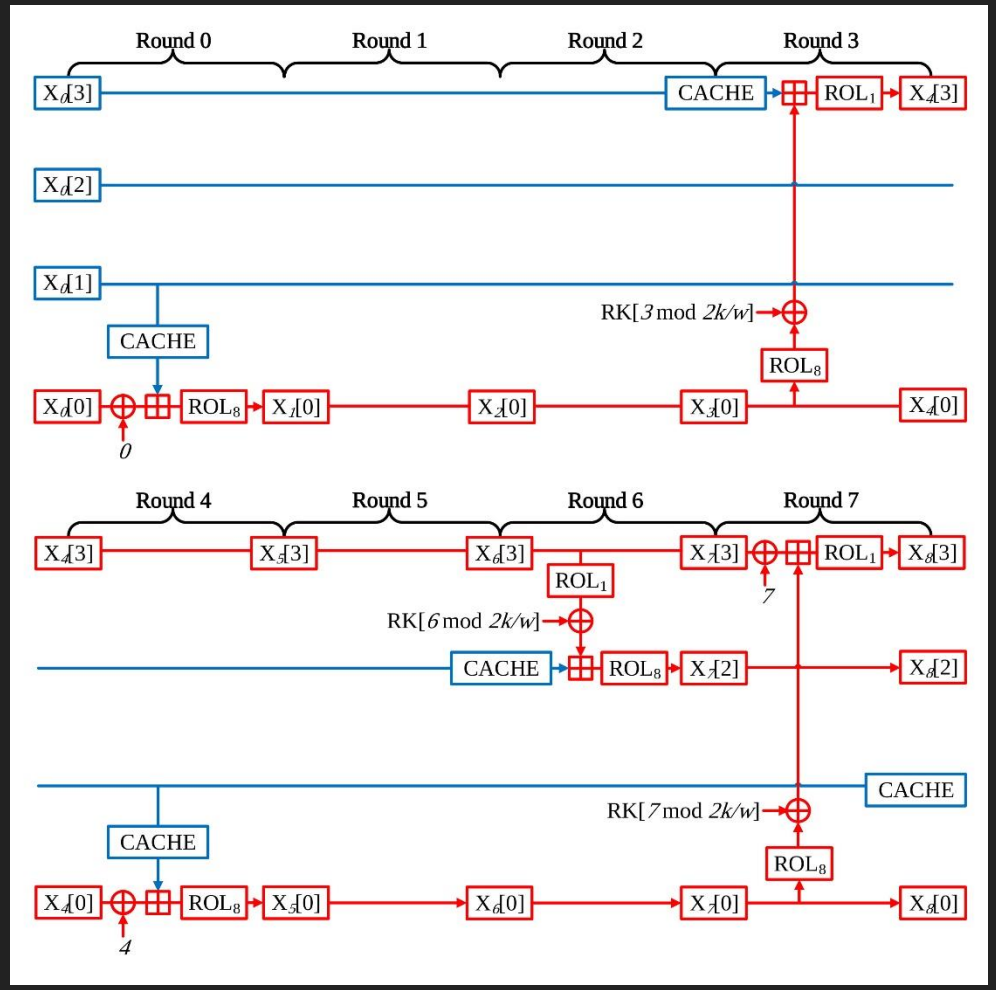

Fig 10. Optimized eight rounds of CHAM block cipher

Our Works: CTR mode optimization

Incase of: CHAM block cipher

- CHAM-64/128 대상으로 추가구현
- 블록 크기가 16-bit인 관계로, 두 가지 버전으로 구현
  - 16-bit 카운터 버전: 128/128, 128/256과 동일한 구현
  - 32-bit 카운터 버전: [그림 11]의 구조
- 32-bit 카운터 버전의 특징
  - 초기 카운터 블록이 2개로 늘어남
    - 16-bit 블록에 32-bit 카운터를 담기 위함
  - 더 많은 구간이 카운터의 영향을 받음

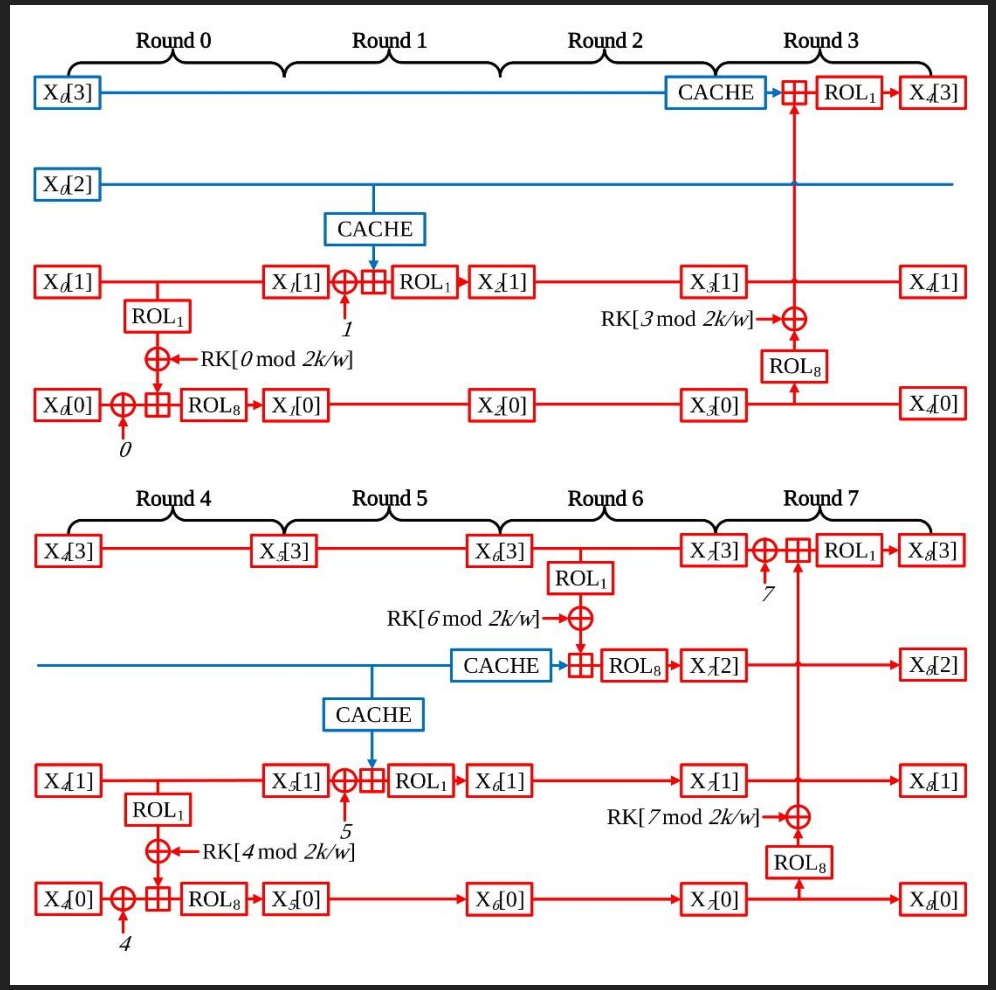

Fig 11. Optimized eight rounds of CHAM-64/128 32-bit counter

Our Works: CTR mode optimization

Incase of: HIGHT block cipher

- HIGHT는 단일 규격만 제공되며 내부 파라미터는 [표 4]를 따름
- 0~3라운드까지 사전연산이 가능

Cipher	n (block size)	k (key size)	rk (round key size)	r (number of rounds)
HIGHT-64/128	64-bit	128-bit	64-bit	32

Table 4. Parameters of HIGHT block cipher

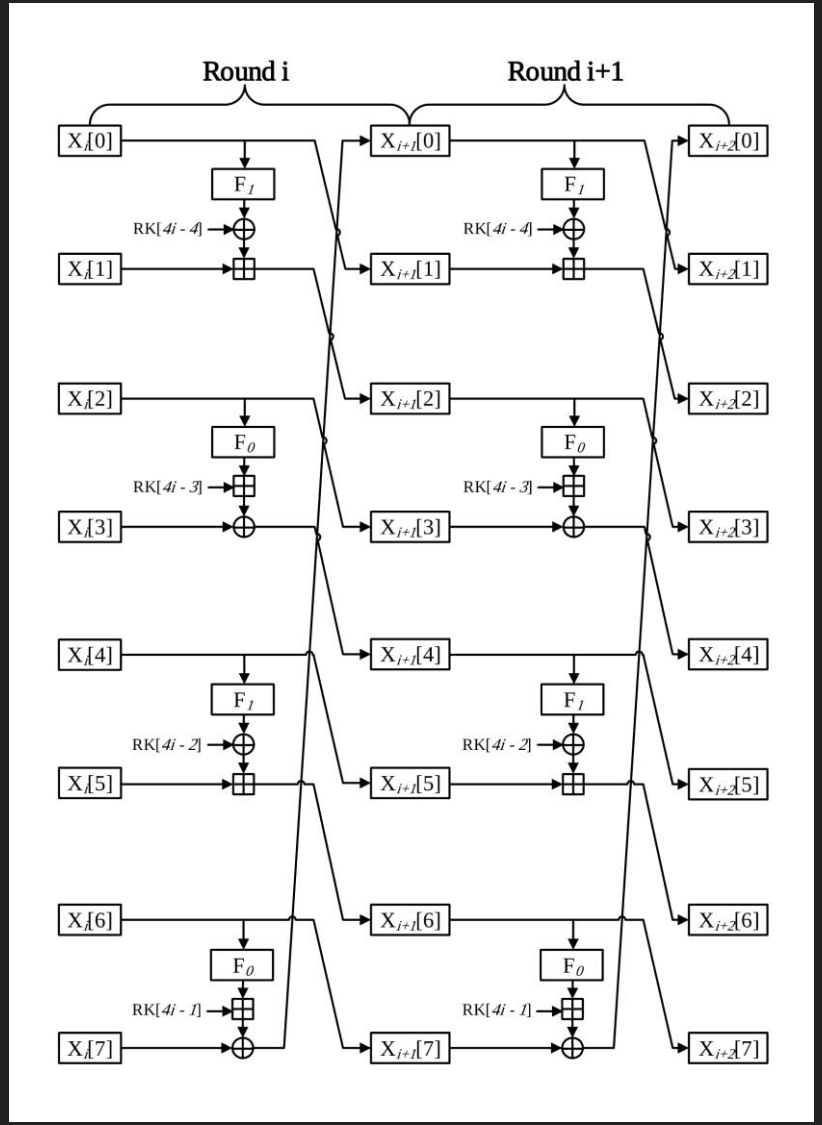

Fig 12. Encryption round function structure of HIGHT block cipher

Our Works: CTR mode optimization

Incase of: HIGHT block cipher

- [그림 13] 형태로 라운드 함수 재구성
- 생략된 명령어
  - ADD: 7개
  - ADC: 5개
  - XOR: 5개
  - MOVW: 5개
  - LD: 5개

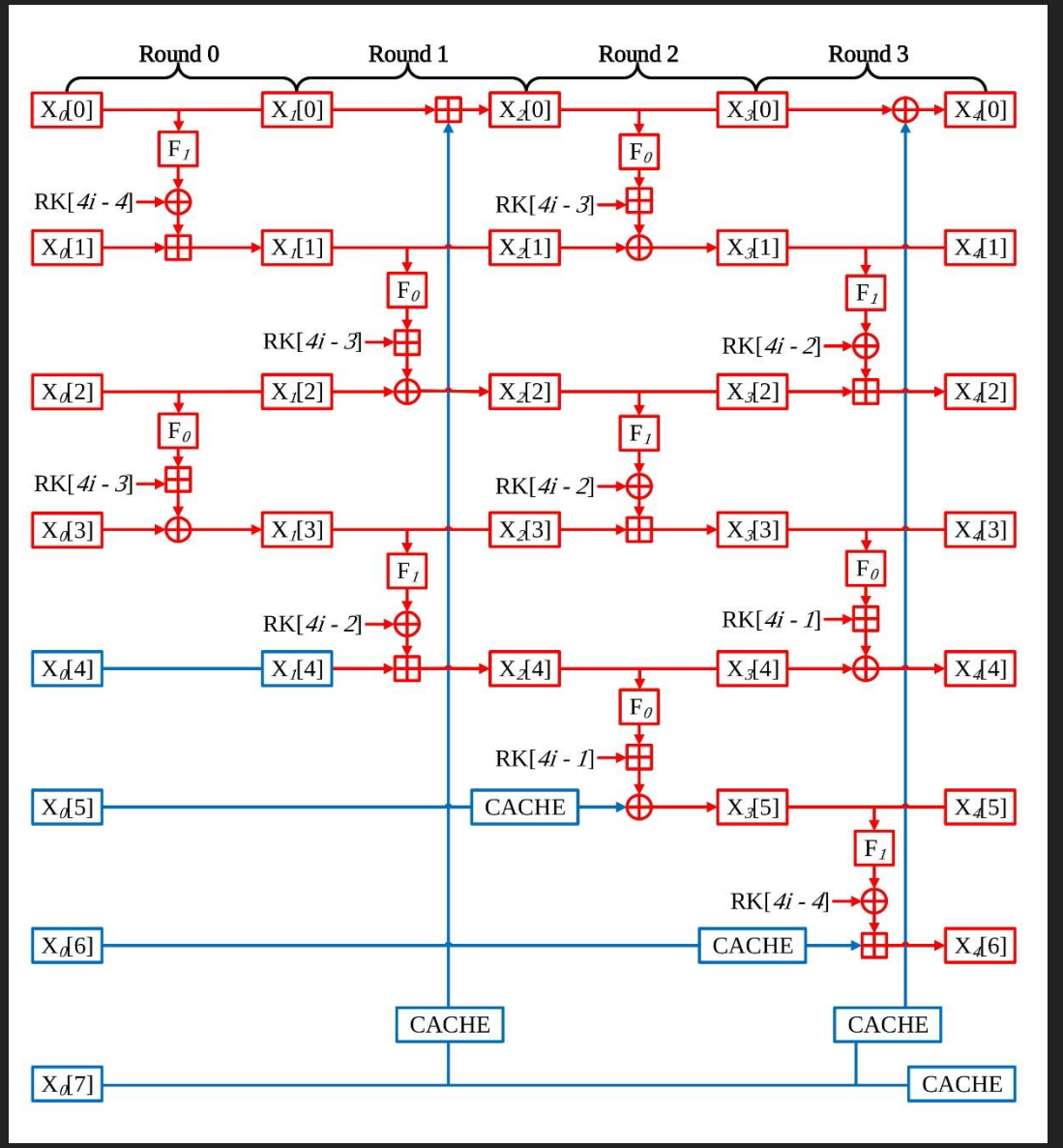

Fig 13. Optimized four rounds of HIGHT block cipher

Our Works: CTR mode optimization

Incase of: LEA block cipher

- LEA는 3가지 규격이 있으며 내부 파라미터는 [표 5]를 따름
- 0~2라운드까지 사전연산이 가능

Cipher	n (block size)	k (key size)	rk (round key size)	r (number of rounds)
LEA-128/128	128-bit	128-bit	192-bit	24
LEA-128/192	128-bit	192-bit	192-bit	28
LEA-128/256	128-bit	256-bit	192-bit	32

Table 5. Parameters of LEA block cipher

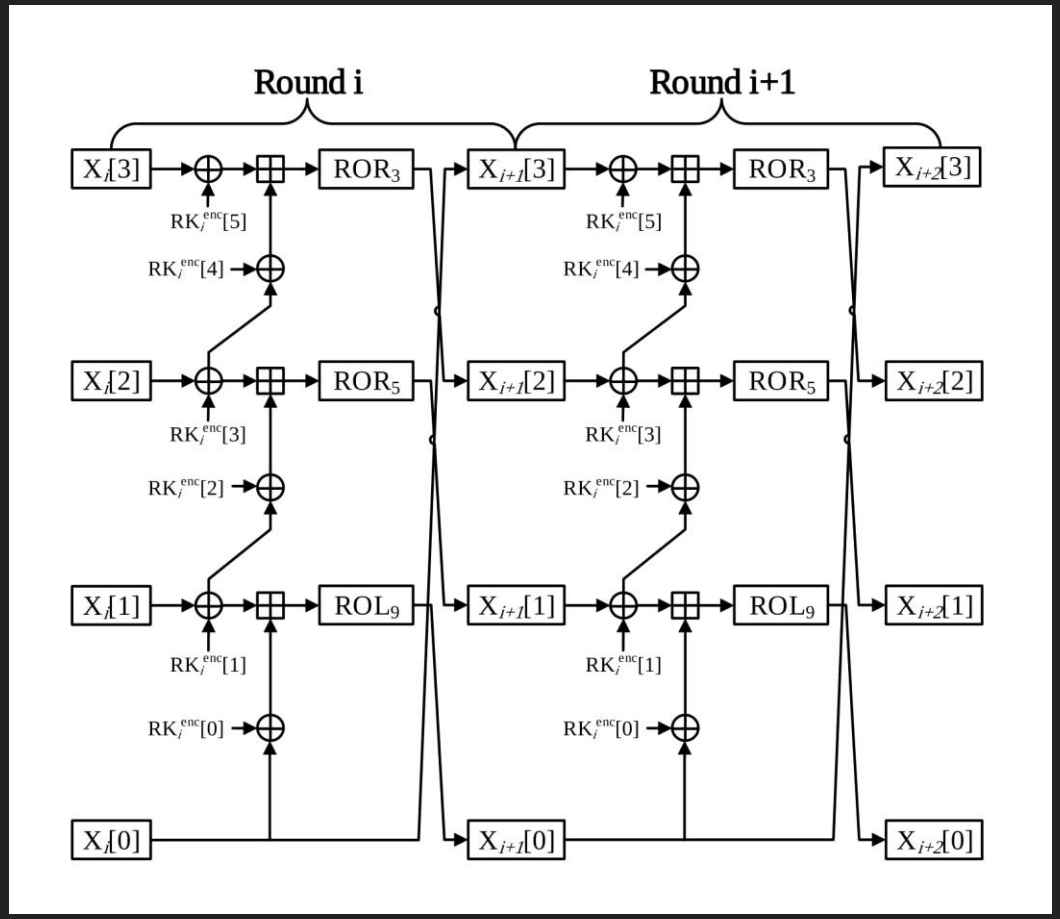

Fig 14. Encryption round function structure of LEA block cipher

Our Works: CTR mode optimization

Incase of: LEA block cipher

- [그림 20] 형태로 라운드 함수 재구성
  - 128과 192,256의 차이는 사용하는 라운드 키의 값
  - 구조 자체는 동일함
- 생략된 명령어, 괄호 안은 LEA-128/192,256 경우
  - LSL: 5개
  - ROL: 15개
  - LSR: 3개
  - ROR: 12개
  - ADD: 4개
  - ADC: 17개
  - XOR: 33개
  - LD: 20개 (44개)
  - CLR: 1개

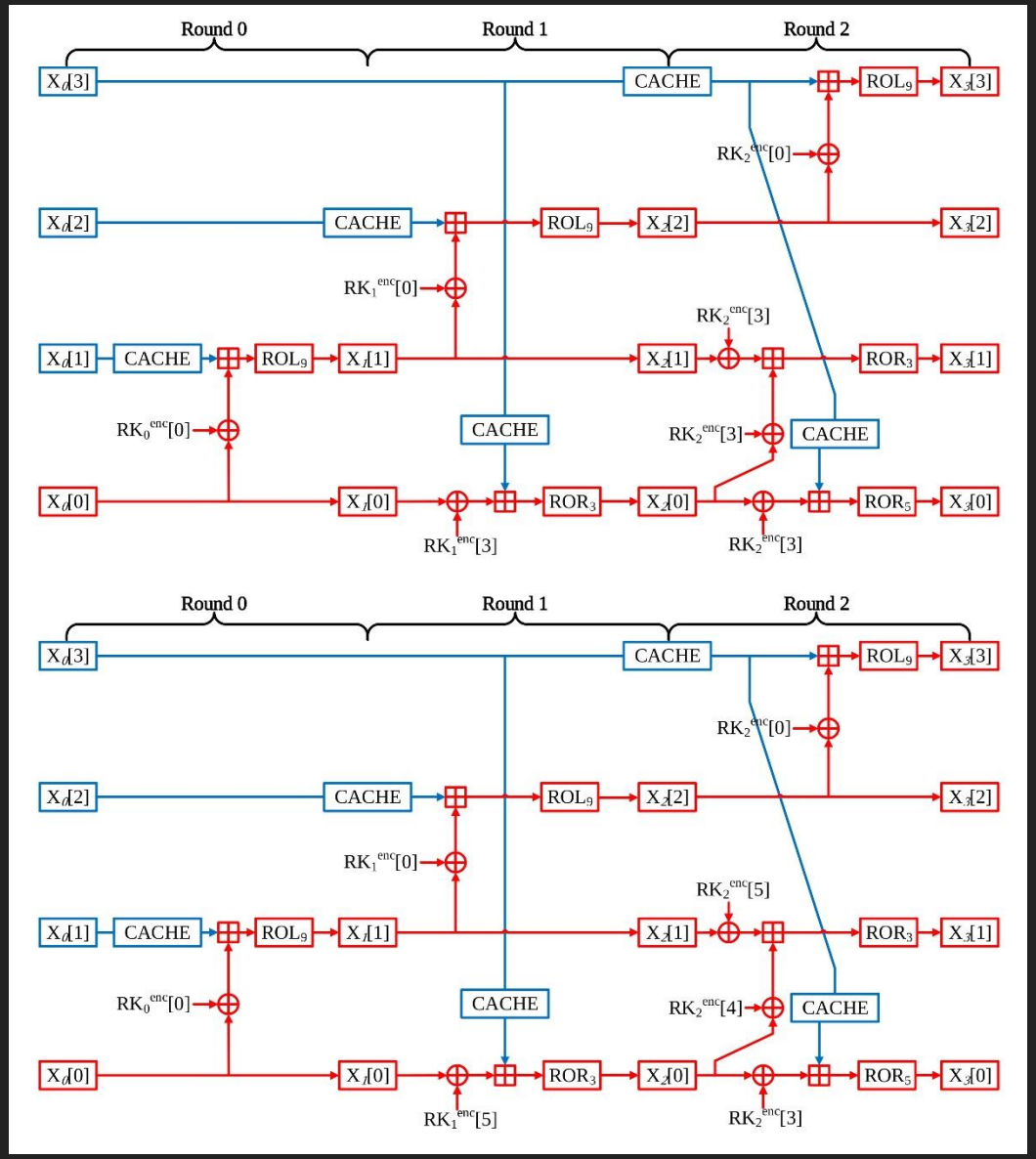

Fig 15. Optimized three rounds of LEA block cipher (Top: LEA-128/128, Bottom: LEA-128/192,256)

Our Works: CTR mode optimization

Incase of: GPU Environment

- GPU의 특성을 사용하여 대량의 데이터에 대한 병렬연산 기법을 적용
  - GPU를 구성하는 스레드들이 각자 하나의 블록을 암호화
  - 카운터 모드를 사용하여 암호화 수행
    - GPU에서 스레드 고유 번호(Thread ID)를 카운터 값으로 사용
    - 논스를 이용한 사전연산 기법 적용 가능

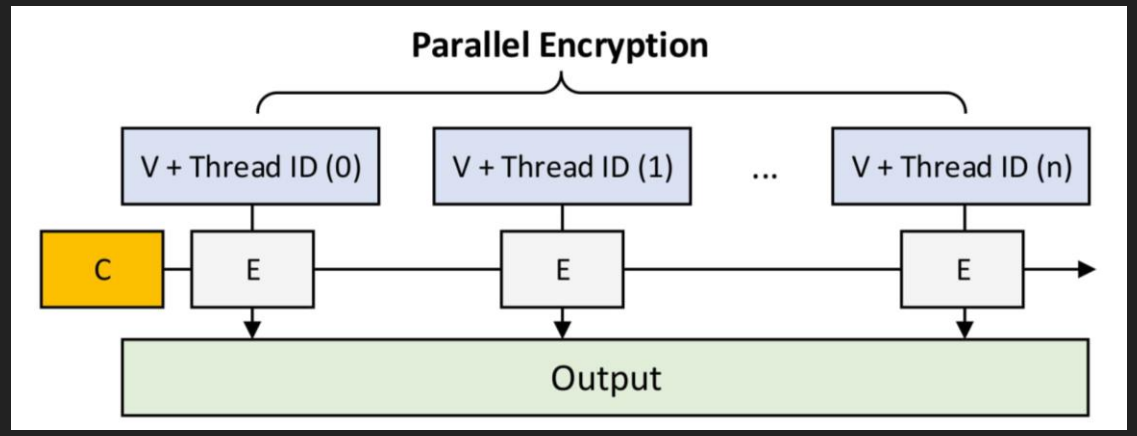

Fig 16. Parallel encryption by each thread in GPU

Our Works: CTR mode optimization

Incase of: GPU Environment

- CPU와 GPU 사이에서 수행되는 메모리 복사 시간 최적화 기법을 적용
  - 카운터 모드를 사용하여 평문 메모리 복사 시간 생략 가능
  - CUDA Stream을 사용하여 비동기 연산(Asynchronous execution) 수행
    - CPU가 쉬지 않고 즉시 다음 작업을 수행함으로써 유휴 시간 감소

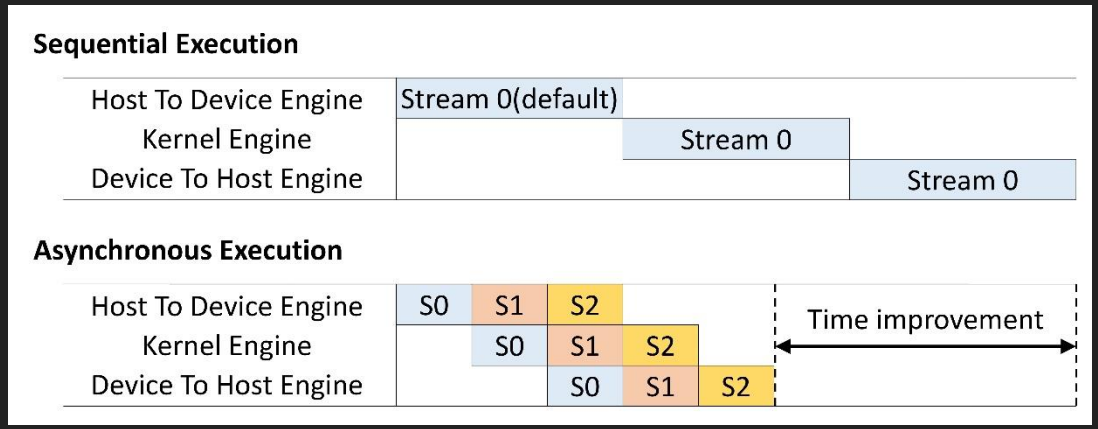

Fig 17. Asynchronous execution technique using CUDA stream

Our Works: CTR mode optimization

Incase of: GPU Environment

- 공유 메모리를 사용하여 GPU 내부 메모리 접근속도 향상
  - 공유 메모리를 사용하면서 발생할 수 있는 문제 제시 및 해결
    - 뱅크 충돌(Bank conflict)
    - 워프 직렬화(Warp divergence)

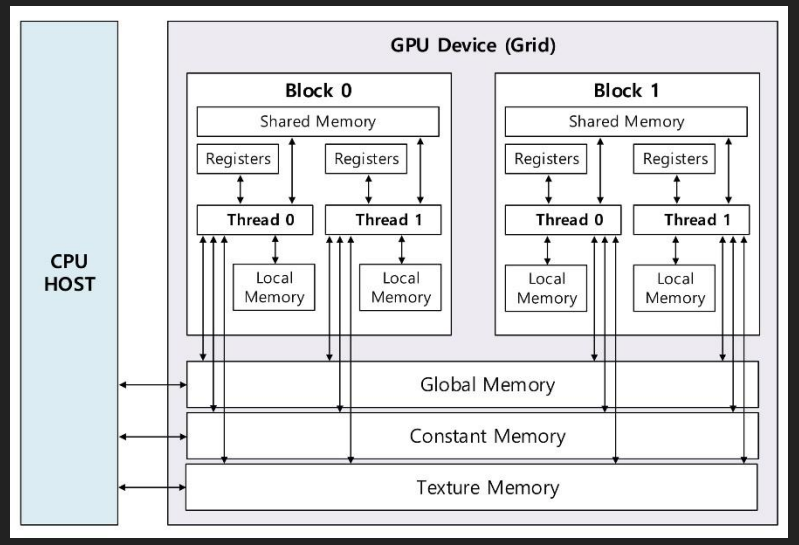

Fig 18. GPU memory construction

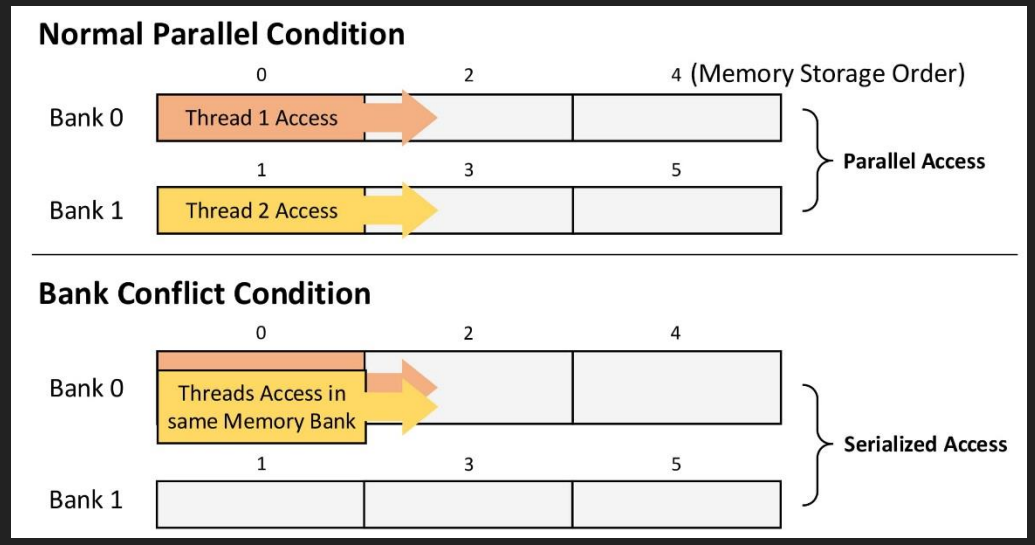

Fig 19. CUDA bank conflict situation

Our Works: CTR mode optimization

Incase of: GPU Environment

- GPU 인라인 어셈블리어(Inline PTX)를 사용하여 효율적인 레지스터 및 연산 사용

```
asm("{\n\t"
        " .reg.type     t0,t1,t2,t3,t4,t5;      \n\t"
        " shf.l.type    t0, %5, %5, 0x1;        \n\t"
        " xor.type      t1, t0, %8;             \n\t"
        " xor.type      t2, %4, 0x0;            \n\t"
        " add.type      t3, t1, t2;             \n\t"
        " shf.l.type    t4, t3, t3, 0x8;        \n\t"
        " shf.l.type    t0, %6, %6, 0x8;        \n\t"
        " xor.type      t1, t0, %9;             \n\t"
        " xor.type      t2, %5, 0x1;            \n\t"
        " add.type      t3, t1, t2;             \n\t"
        " shf.l.type    t5, t3, t3, 0x1;        }\n\t"
      : "+r"(x), "+r"(y), "+r"(z), "+r"(w)
      : "r"(x), "r"(y), "r"(z), "r"(w),
      "r"(rk[0]), "r"(rk[1]), "r"(rk[2]), "r"(rk[3]),
      "r"(rk[4]), "r"(rk[5]), "r"(rk[6]), "r"(rk[7])
);
```

Fig 20. Inline PTX code of CHAM

```
asm("{\n\t"
        " .reg.type     t                       \n\t"
        " mov.type      t, %9;                  \n\t"
        " ld.shared.type    t, array0[t];       \n\t"
        " add.type      t, t, %19;              \n\t"
        " xor.type      %0, t, %8;              \n\t"
        " mov.type      t, %11;                 \n\t"
        " ld.shared.type    t, array1[t];       \n\t"
        " add.type      t, t, %18;              \n\t"
        " xor.type      %2, t, %10;             \n\t"
        " mov.type      t, %13;                 \n\t"
        " ld.shared.type    t, array0[t];       \n\t"
        " add.type      t, t, %17;              \n\t"
        " xor.type      %4, t, %12;             \n\t"
        " mov.type      t, %15;                 \n\t"
        " ld.shared.type    t, array1[t];       \n\t"
        " add.type      t, t, %16;              \n\t"
        " xor.type      %6, t, %14;             }\n\t"
      " : "+r"(XX[0]), "+r"(XX[1]), ... , "+r"(XX[7])
      : "r"(XX[0]), "r"(XX[1]), ... , "r"(XX[7]),
      "r"(rk[i*4]), "r"(rk[i*4 + 1]), "r"(rk[i*4 + 2]), "r"(rk[i*4 + 3])
);
```

Fig 21. Inline PTX code of HIGHT

```
asm("{\n\t"
        " xor.type      %2, %6, %12;            \n\t"
        " xor.type      %3, %7, %13;            \n\t"
        " add.type      %3, %6, %7;             \n\t"
        " shf.r.type    %3, %7, %7, 0x3         \n\t"
        " xor.type      %1, %5, %10;            \n\t"
        " xor.type      %2, %6, %11;            \n\t"
        " add.type      %2, %5, %6;             \n\t"
        " shf.r.type    %2, %6, %6, 0x5;        \n\t"
        " xor.type      %0, %4, %8;             \n\t"
        " xor.type      %1, %5, %9;             \n\t"
        " add.type      %1, %4, %5;             \n\t"
        " shf.l.type    %1, %5, %5, 0x9;        \n\t"
        " xor.type      %3, %7, %14;            \n\t"
        " xor.type      %0, %4, %15;            \n\t"
        " add.type      %0, %7, %4;             \n\t"
        " shf.r.type    %0, %4, %4, 0x3;        }\n\t"
      : "+r"(W), "+r"(X), "+r"(Y), "+r"(Z)
      : "r"(W), "r"(X), "r"(Y), "r"(Z),
      "r"(rk[i]), "r"(rk[i + 1]), "r"(rk[i + 2]), "r"(rk[i + 3]),
      "r"(rk[i + 4]),"r"(rk[i + 5]),"r"(rk[i + 10]), "r"(rk[i + 11])
);
```

Fig 22. Inline PTX code of LEA

Our Works: CTR_DRBG optimization

- 내부상태의 C와 V가 초기 Zero임을 이용하여 사전연산 기법을 적용
- 인스턴스 생성함수의 유도함수에서 입력 값의 초기 128-bit가 Constant (상수) 임을 이용
- 인스턴스 생성함수의 내부 갱신함수에서 내부상태가 Zero임을 이용
- 이를 사전연산 테이블(Look Up Table) 형태로 저장가능
- 초기, CTR_DRBG에서 인스턴스 생성함수가 호출될 때 사전연산한 LUT 호출
- LUT를 이용하면 인스턴스 생성함수에서 최소 4번, 최대 6번의 암호화 과정을 생략 가능함.
- 출력함수의 CTR mode 암호화 과정에서 본 논문에서 제안한 블록암호 사전연산 기법 적용
- 출력함수 호출 시 처음 암호화때 테이블을 생성하고 그 뒤로는 고정키 시나리오로 최적화 가능

Our Works: CTR_DRBG optimization

Incase of: Derivation Function

- 인스턴스 생성함수에서 호출되는 유도함수
  - S는 유도함수에서 사용되는 데이터
  - 초기 128bit인 C와 0(pad) 는 상수
- 이를 이용하여 사전연산 테이블을 생성
  - LEA-128, CHAM-128/128의 경우
    2번의 암호화 과정을 생략
  - 그 외의 경우 3번의 암호화 과정을 생략
- 리시드 함수 (Reseed Function)에서
  호출되는 유도함수에도 적용가능
- 사전연산 테이블은 Constant로 갱신이 필요 없음

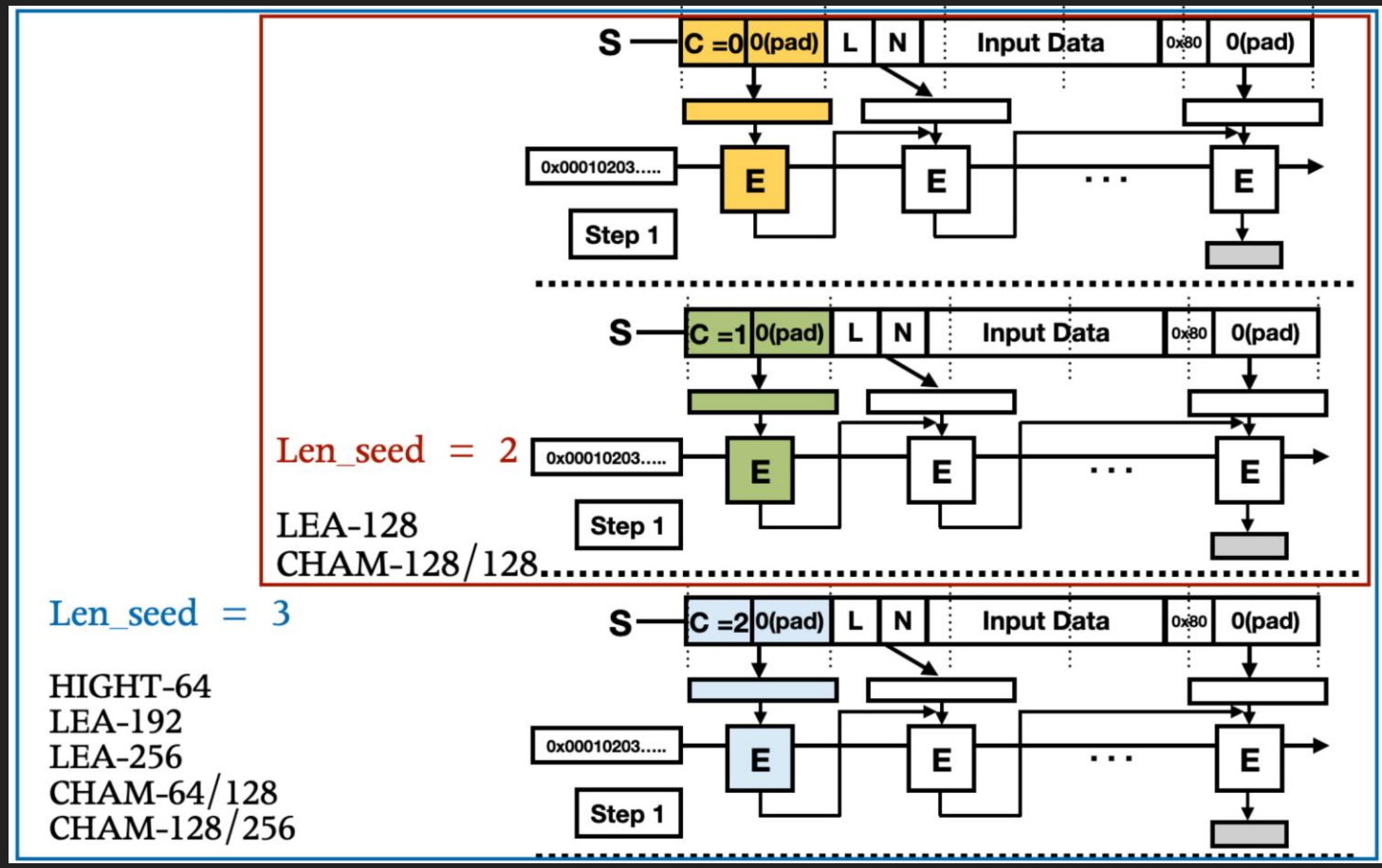

Fig 23. Optimized Derivation Function of Instantiate Function

Our Works: CTR_DRBG optimization

Incase of: Update Function

- 인스턴스 생성함수에서 호출되는 내부갱신 함수
  - 내부 갱신함수의 입력 값은 Zero임을 이용
- 이를 이용하여 사전연산 테이블을 생성
  - LEA-128, CHAM-128/128의 경우
    2번의 암호화 과정을 생략
  - 그 외의 경우 3번의 암호화 과정을 생략
- 사전연산 테이블은 Constant로 갱신이 필요 없음

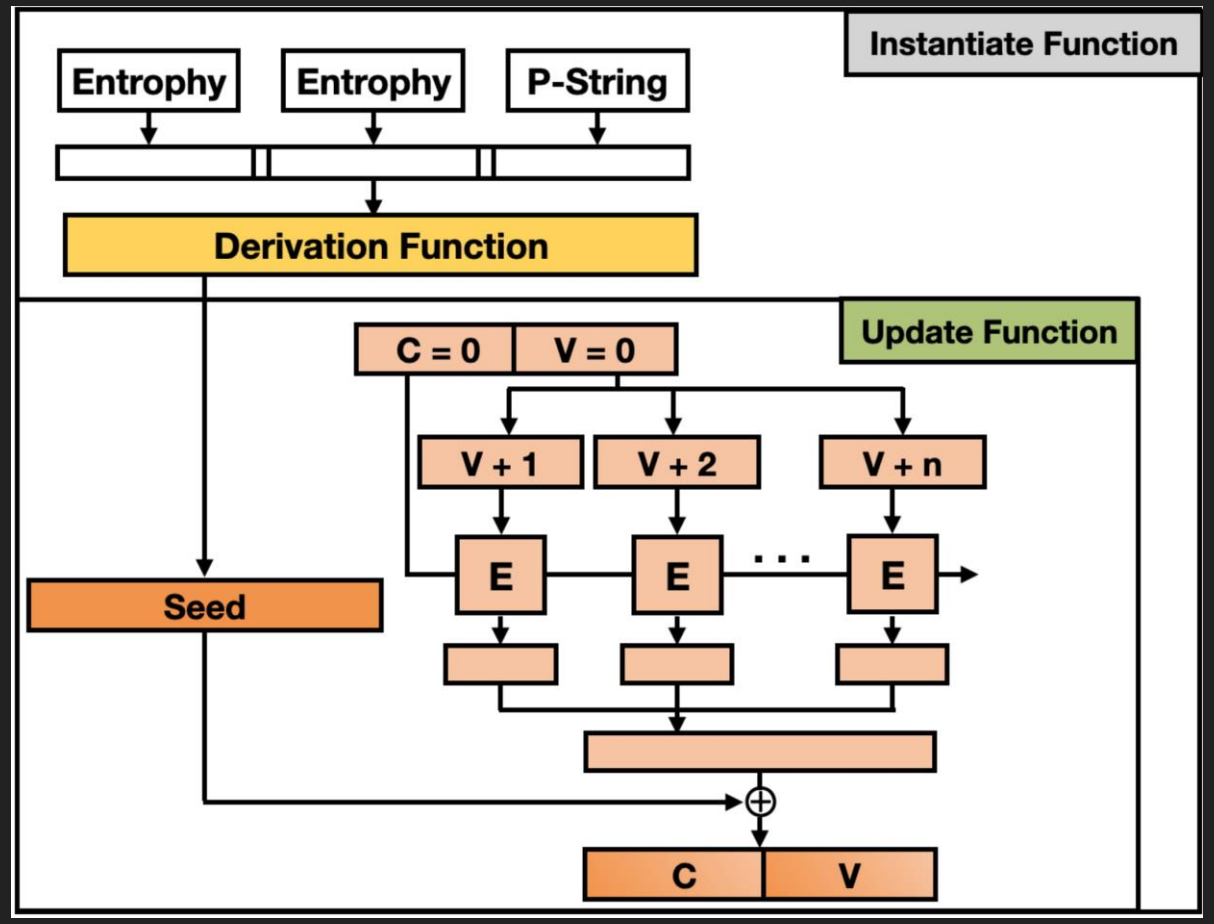

Fig 24. Optimized Update Function of Instantiate Function

Our Works: CTR_DRBG optimization

Incase of: Extract Function

- 추출생성함수에서 호출되는 출력함수
  - 출력함수는 CTR mode 기반으로 난수 추출
  - 본 논문에서 제안한 CHAM,LEA,HIGHT의 최적화 이용
    - CTR mode 최적화를 이용

- V+1: 암호화 과정 중에 사전연산 테이블 생성
- V+2~n: 사전 연산 테이블을 이용하여 최적화
  - V+1에서 생성한 LUT를 이용하여
    암호화 과정 중 일부 생략

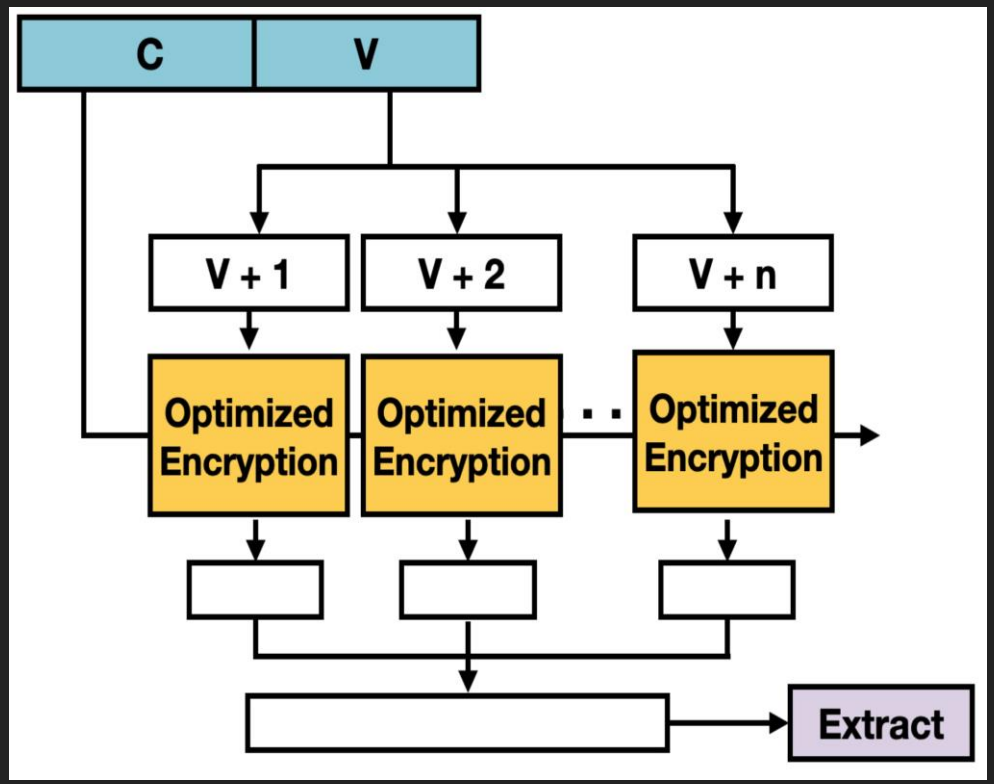

Fig 25. Optimized Extract Function of Instantiate Function

Our Works: CTR_DRBG optimization

Incase of: GPU Environment

- GPU로 CTR_DRBG 주요 연산 고속화
  - 카운터 모드를 사용하여 연산하는 함수
    - 내부 갱신 함수(Update function)
      - 2~3 스레드가 각 블록 암호화 수행
    - 출력 함수(Extract function)
      - 각 스레드가 병렬 암호화 수행

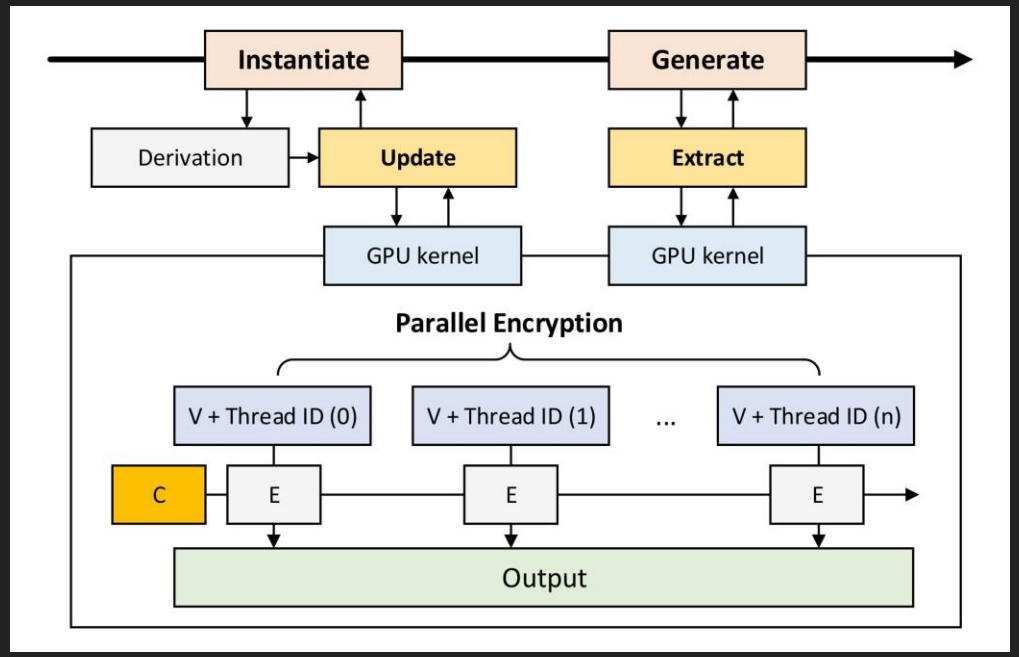

Fig 26. Parallel encryption structure for extract function

Our Works: Evaluation

- CTR 최적 구현 환경
  - 8-bit AVR Microcontroller: ATmega 128
    - 128KB FLASH memory
    - 4KB SRAM
    - 32개의 범용 레지스터
- CTR 최적화 평가 모델
  - Separated model: 사전연산과 암호화 과정이 분리된 모델
    - 이 모델은 암호화를 진행하지 않으므로, 실제로 암호화를 진행하는 Using cache table model과 평균치를 계산
  - Pre-computation in online: 사전연산을 하며 블록 1개를 암호화하는 모델
  - Using cache table model: 사전연산 된 값을 활용하여 모든 블록을 암호화하는 모델
- 비교 대상
  - 기존 최적 구현 중에서, 가장 뛰어난 최적 구현 결과와 비교
  - 기존 최적 구현이 없다면, 알고리즘의 최초 구현 성능과 비교

Our Works: Evaluation

Incase of: CHAM Function

c: counter mode,
b: 16-bit counter,
s: separated model,
o: in online model

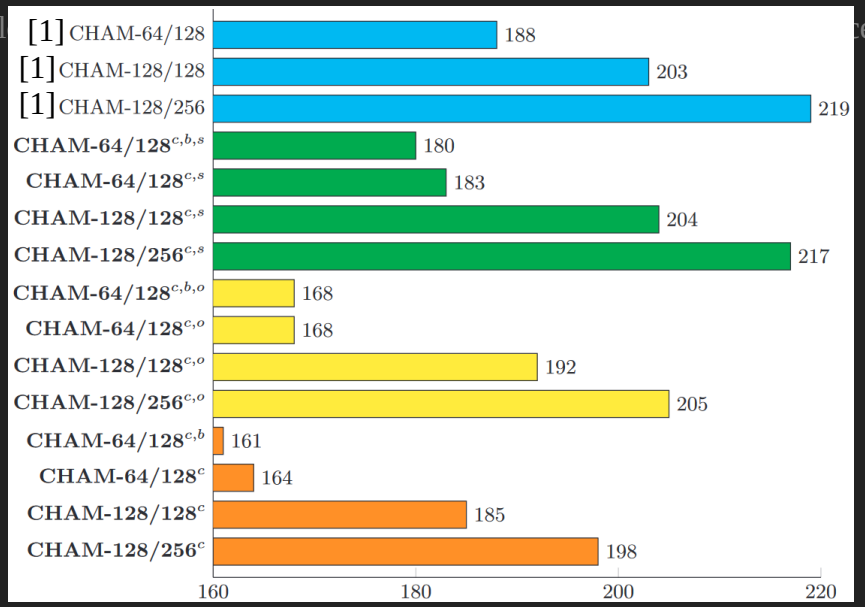

Fig 27. Comparison of execution time for CHAM implementations

- [1]과 비교 결과, [표 6] 만큼 성능 향상
- 사전연산을 활용하면 최대 12.8%의 성능 향상
  - 규격 외 구현인 16-bit 카운터 구현을 제외한 결과
- 분리 모델에 비해 동시진행 모델의 성능이 높음
  - 평균 값을 고려해도 동시진행 모델이 성능이 뛰어남
  - 첫번째 블록 암호화에 드는 시간이 성능 차이에 큰 역할

Cipher	Separated model	Pre-computation in online	Using cache table model
CHAM-64/128 (16-bit)	4.3%	10.6%	16.8%
CHAM-64/128	2.7%	10.6%	12.8%
CHAM-128/128	0.0%	5.4%	8.9%
CHAM-128/256	1.0%	6.4%	9.6%

Table 6. Comparison result of CHAM

[1] D. Roh, B. Koo, Y. Jung, I. W. Jeong, D.-G. Lee, D. Kwon, and W.-H. Kim. Re-vised version of block cipher CHAM. In International Conference on Information Security and Cryptology, pages 1–19. Springer, 2019.

Our Works: Evaluation

Incase of: HIGHT Function

- [3]과 비교 결과, [표 7] 만큼 성능 향상
  - [2]: 최초 구현
  - [3]: 기존 최적 구현
- 사전연산을 활용하면 최대 3.8%의 성능 향상
- CHAM 구현과 마찬가지로 동시진행 모델의 성능이 뛰어남
- CHAM에 비해 낮은 성능 향상 폭
  - CHAM은 많은 구간에서 사전연산이 가능
  - CHAM은 기존 최적 구현 결과물이 없음
  - 따라서 HIGHT 최적 구현물이 성능이 나쁘다고 할 수 없음

c: counter mode,
s: separated model,
o: in online model

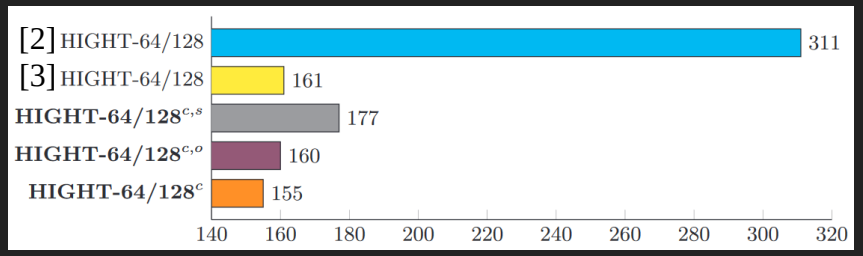

Fig 28. Comparison of execution time for HIGHT implementations

Cipher	Separated model	Pre-computation in online	Using cache table model
HIGHT-64/128	-9.9%	0.7%	3.8%

Table 7. Comparison result of HIGHT

[2] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, et al. HIGHT: A new block cipher suitable for low-resource device. In International Workshop on Cryptographic Hardware and Embedded Systems, pages46–59. Springer, 2006.
[3] H. Seo, I. Jeong, J. Lee, and W.-H. Kim. Compact implementations of ARX-based block ciphers on IoT processors. ACM Transactions on Embedded Computing Systems (TECS), 17(3):1–16, 2018.

Our Works: Evaluation

Incase of: LEA Function

c: counter mode,
s: separated model,
o: in online model

- [3], [5]와, [표 8] 만큼 성능 향상
  - [4]: 최초 구현
  - [3]: LEA-128/128 최적 구현
  - [5]: LEA-128/192, LEA-128/256 최적 구현
- 사전연산을 활용하면 최대 5.9%의 성능 향상
- LEA 구현물도 동시진행 모델의 성능이 뛰어남
- 기존 최적 구현물이 많이 존재
  - 사전연산 활용 시 기존 최적 구현물보다 뛰어난 성능

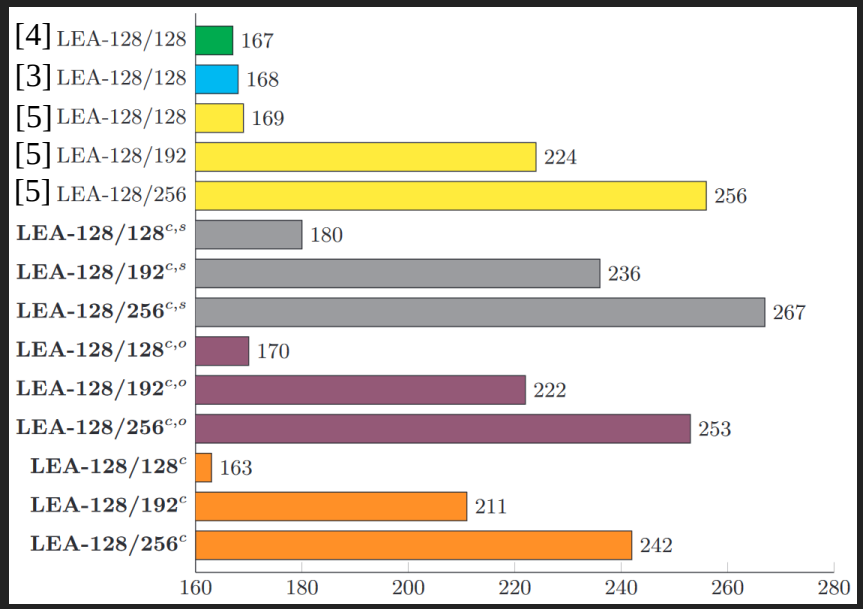

Fig 29. Comparison of execution time for LEA implementations

Cipher	Separated model	Pre-computation in online	Using cache table model
LEA-128/128	-7.3%	-0.1%	3.6%
LEA-128/192	-5.1%	0.1%	5.9%
LEA-128/256	-4.1%	1.2%	5.5%

Table 8. Comparison result of LEA

[4] H. Seo, K. An, and H. Kwon. Compact LEA and HIGHT implementations on8-bit AVR and 16-bit MSP processors. In International Workshop on Information Security Applications, pages 253–265. Springer, 2018.
[5] H. Seo, Z. Liu, J. Choi, T. Park, and H. Kim. Compact implementations of LEA block cipher for low-end microprocessors. In International Workshop on Information Security Applications, pages 28–40. Springer, 2015.

Our Works: Evaluation
Incase of: GPU Environment

- GPU 실험 환경
  - 암호화 데이터 크기는 최소 256 MB에서 최대 1024 MB까지 가변
  - 병렬 연산에 사용한 GPU 블록 당 스레드 수는 최소 256개에서 최대 1024개까지 가변
  - 성능 결과는 가장 성능이 뛰어난 데이터 크기와 GPU 블록 당 스레드 수 기준으로 제시
  - 테스트는 각각 1000번 반복 후 평균값 계산
  - 성능 측정 결과는 모두 CPU-GPU 사이 메모리 복사 시간이 포함된 시간 기준으로 측정

환경	이름
CPU	AMD Ryzen 5 3600
GPU	NVIDIA RTX 2070
GPU 아키텍처	튜링(Turing)
GPU 코어 수	2,340
GPU 메모리 크기	8 GB
CUDA 버전	10.2
운영체제	윈도우 10 (Windows 10)

Table 9. GPU implementation test environment

Our Works: Evaluation
Incase of: GPU Environment

- [표 10]은 GPU 환경에서 초당 암호화 처리량(Throughput)을 기록
- 라운드 키 개수가 제일 적은 CHAM-128/128이 최고 성능을 보임

Block Cipher	CHAM-64/128	CHAM-128/128	CHAM-128/256	LEA-128	LEA-192	LEA-256	HIGHT-64/128
ECB 모드 병렬 구현	2,133	2,171	2,184	2,169	2,115	2,122	2,047
CTR 모드 병렬 구현	2,976	3,324	3,368	3,239	3,240	3,121	2,948
CTR 모드 CUDA Stream 최적화 구현	3,882	4,253	4,176	3,412	3,365	3,356	3,419
ECB 모드 병렬 구현 대비 최적화 성능 향상치	81%	95%	91%	57%	59%	58%	67%

Table 10. Comparison of throughput (MB/s) for CHAM, LEA, and HIGHT implementations on GPU

Our Works: Evaluation

Incase of: CTR_DRBG in 8-bit AVR Microcontroller

- [표 11] 은 8-bit AVR 환경에서 추출 난수 byte를 기준으로 최고 성능향상 수치를 기록
- CHAM과 HIGHT의 경우 32byte 난수를 추출할 때 최고의 성능을 보임
  - 유도함수와 내부 갱신 함수의 최적화 성능 향상 비가 출력함수의 성능 향상 비보다 높음
- LEA의 경우 출력 난수열의 길이가 증가할 수록 성능향상 비가 증가
  - 출력함수의 성능향상 비가 유도함수와 내부 갱신 함수 보다 높음

Block Cipher	CHAM-64/128	CHAM-64/128 (32)	CHAM-128/128	CHAM-128/256
Extracted Byte	32Byte	32Byte	32Byte	32Byte
CTR_DRBG	15.3%	14.3%	16.6%	21.1%
Block Cipher	**LEA-128/128**	**LEA-128/192**	**LEA-128/256**	**HIGHT-64/128**
Extracted Byte	1024Byte	1024Byte	1024Byte	32Byte
CTR_DRBG	26.7%	36.2%	37.2%	8.7%

Table 11. Performance Comparison of CTR_DRBG to the previous best results on AVR.
The result is based on the number of extracted random numbers

Our Works: Evaluation

Incase of: CTR_DRBG in GPU Environment

- [그림 30]의 세로축은 GPU 환경에서 초당 난수 출력량 (Throughput)을 기록
- 가로축은 CTR_DRBG 출력 난수 길이를 의미
- 출력 난수 길이가 길어질수록 성능 향상
- 난수 출력 길이가 128MB일 때 CHAM-128/128이 최고 성능을 보임

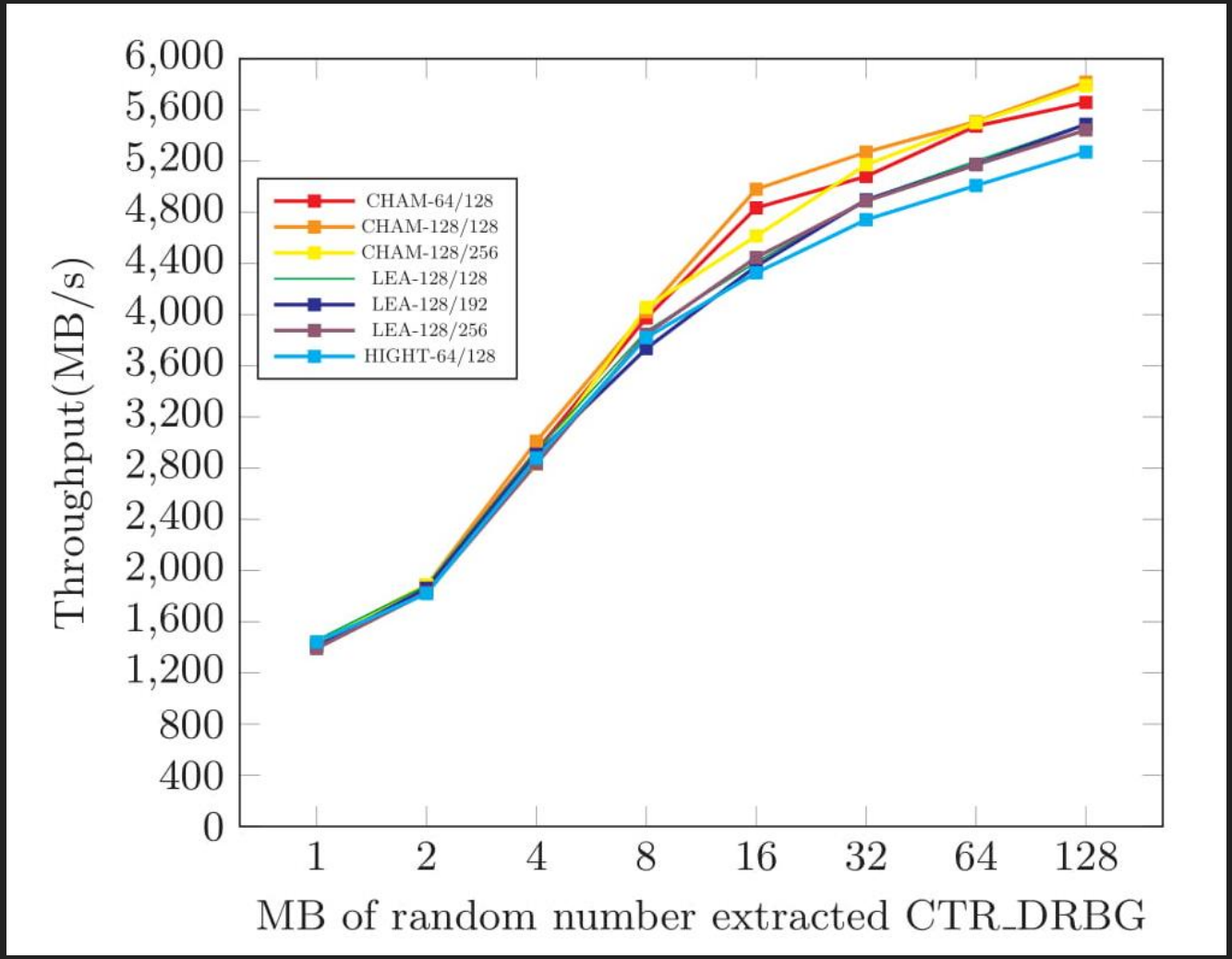

Fig 30. Throughput (MB/s) for CTR_DRBG implementations on GPU

Conclusion

ARX-based Korean Block Ciphers with CTR Mode and CTR_DRBG on Embedded Processors and GPU

국 가 암 호 공 모 전

Conclusion: Contribution

- 8-bit AVR 마이크로컨트롤러를 대상으로 국산 블록암호 최적 구현을 시도
  - CHAM, HIGHT, LEA 알고리즘 구현
- 사전연산을 하여 일부 라운드의 연산을 생략
- 3 종류의 암호 알고리즘에서 모두 기존보다 뛰어난 성능을 보임
  - CHAM: 12.8%, HIGHT: 3.8%, LEA: 5.9%
- GPU를 이용한 대용량 데이터 암호화 성능 고속화
  - IoT 기기를 관리하는 서버에서 다수의 기기들에 대한 신속한 암/복호화 기능 제공
  - 클라우드 컴퓨팅 서비스를 제공하는 서버에서 대용량 파일에 대한 신속한 암/복호화 기능 제공
- GPU를 이용한 난수 생성기의 난수 출력 고속화
  - 난수를 사용하는 다양한 암호 시스템에 신속하게 난수 제공
  - 출력된 난수는 크기가 큰 난수를 필요로 하는 공개키 암호 및 양자 내성 암호에 사용

Conclusion: Future works

- 본 논문에서 제안한 알고리즘은 소형 IoT 기기 등 극한 환경에 최적화된 기법
  - 실제 적용 후 효율성을 테스트
- 제안한 기법은 블록암호에서 손쉽게 적용이 가능
  - ARX 기반 블록암호의 경우 더욱 용이함
- 8-bit AVR 외에 다른 환경에서 추가적인 구현
  - 16-bit MSP, 32-bit ARM
- 국내 경량 블록 암호 뿐만 아니라 해외 경량 암호에 대한 최적화 연구 예정
  - 해외 경량 블록 암호 알고리즘 GIFT, PRESENT
  - NIST 경량 암호(Lightweight Cryptography) 라운드 2 경쟁 알고리즘

THANK YOU

국 가 암 호 공 모 전

Appendix

[1] D. Roh, B. Koo, Y. Jung, I. W. Jeong, D.-G. Lee, D. Kwon, and W.-H. Kim. Re-vised version of block cipher CHAM. In International Conference on Information Security and Cryptology, pages 1–19. Springer, 2019.
[2] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, et al. HIGHT: A new block cipher suitable for low-resource device. In International Workshop on Cryptographic Hardware and Embedded Systems, pages46–59. Springer, 2006.
[3] H. Seo, I. Jeong, J. Lee, and W.-H. Kim. Compact implementations of ARX-based block ciphers on IoT processors. ACM Transactions on Embedded Computing Systems (TECS), 17(3):1–16, 2018.
[4] H. Seo, K. An, and H. Kwon. Compact LEA and HIGHT implementations on8-bit AVR and 16-bit MSP processors. In International Workshop on Information Security Applications, pages 253–265. Springer, 2018.
[5] H. Seo, Z. Liu, J. Choi, T. Park, and H. Kim. Compact implementations of LEA block cipher for low-end microprocessors. In International Workshop on Information Security Applications, pages 28–40. Springer, 2015.