Pilsung 블록 암호 및 구현

https://youtu.be/rjRwCG0ekj8

1. Pilsung 블록 암호

- Pilsung은 북한에서 개발한 붉은별 3.0 OS에서 볼 수 있는 블록 암호

- AES 기반의 암호로 라운드 함수와 파라미터가 동일하다.
  - 라운드함수 : SubByte, ShiftRows, MixColumns, AddRoundkey
  - 블록 길이 : 128-bit, 키 길이 : 256-bit, 라운드 수 : 10라운드

- 2020년 CHES 논문 “Cache vs. key-dependency : Side channeling an implementation of Pilsung”

- 블로그 “https://www.kryptoslogic.com/blog/2018/07/a-brief-look-at-north-korean-cryptography/”

1. Pilsung 블록 암호

- AES 블록 암호와의 차이점

	AES 블록 암호	Pilsung 블록 암호
키 확장 알고리즘	-	입력된 키를 해시 값(SHA1)으로 변환 된 160-bit 값을 기존의 AES 키 확장 알고리즘을 통해 라운드 키를 생성
Sbox 테이블 크기	8-bit × 256 = 256-byte	10 rounds ×16-byte × 256-byte = 40960-byte
Shiftrows 순열	고정된 하나의 순열	각 라운드마다 임의의 순열

2. 코드 분석 – 키 확장 및 테이블 확장

```c
void pilsung_set_key(pilsung_ctx * ctx, const uint8_t * k, size_t klen) {
    memset(ctx, 0, sizeof(*ctx));
    pilsung_expand_key(k, klen, ctx);
    gen_enc_perm(ctx);
}
```

```c
void pilsung_expand_key(const uint8_t * k, size_t klen, pilsung_ctx * ctx) {
    uint8_t derived[32] = {0};
    pilsung_shakey(derived, k, klen, g_CryptoKeyLen);
    pilsung_expand_roundkey(derived, ctx->e_key, g_round_cnt);
}
```

```c
void gen_enc_perm(pilsung_ctx * ctx) {
    InitVar(ctx);
    Get_VSboxAll(ctx);
    Get_VPboxAll(ctx);
}
```

```c
typedef struct pilsung_ctx {
    uint8_t sbox_xor_constant; // = 3
    uint8_t sboxes[11][4][4][256]; // S-boxes
    uint8_t pboxes[11][16]; // P-boxes
    uint8_t current_permutation_8[8]; // used for temporary storage
    uint8_t e_key[12 * 16]; // AES scheduled key
} pilsung_ctx;
```

2. 코드 분석 – 암호화

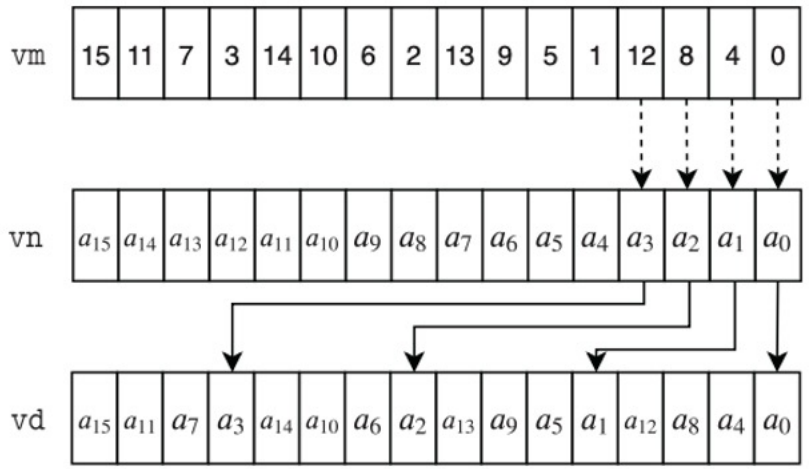
```
void pilsung_encrypt(const pilsung_ctx * ctx, uint8_t output[16], const uint8_t input[16]) {
    block_t block;

    for(size_t i = 0; i < 4; ++i)
        for(size_t j = 0; j < 4; ++j)
            block[i][j] = input[i * 4 + j];

    AddRoundKey(block, (const uint8_t (*)[4])&ctx->e_key[0]);
    for(size_t round = 1; round < 10; ++round) {
        SubBytes(ctx, round, block);
        ShiftRows(ctx, round, block);
        MixColumns(block);
        AddRoundKey(block, (const uint8_t (*)[4])&ctx->e_key[16 * round]);
    }
    SubBytes(ctx, 10, block);
    ShiftRows(ctx, 10, block);
    // No MixColumns
    AddRoundKey(block, (const uint8_t (*)[4])&ctx->e_key[16 * 10]);

    for(size_t i = 0; i < 4; ++i)
        for(size_t j = 0; j < 4; ++j)
            output[i * 4 + j] = block[i][j];
}
```

3. ARM 에서의 구현

- ShiftRows 함수, MixColumns 함수
  - MixColumns 구현
    - ARMv8에서 제공하는 암호를 위한 명령어 사용

AESD	AES single round decryption
AESE	AES single round encryption
AESIMC	AES inverse mix columns
AESMC	AES mix columns

  - ShiftRows 함수
    - TBL 명령어 활용



(b) `tbl vd, {vn}, vm`. The permutation pattern is held in `vm`.

```
ShiftRows:
_ShiftRows:
//-----------------------

ld1.16b {v0}, [x1]
ld1.16b {v1}, [x0]

tbl.16b v0, {v0}, v1

st1.16b {v0}, [x1]

//-----------------------
ret
```

3. ARM 최적화 성능

- Mixcolumns 최적화와 Shiftrows 최적화를 각각 돌렸을 때의 성능
  - 오히려 속도가 느려진다.

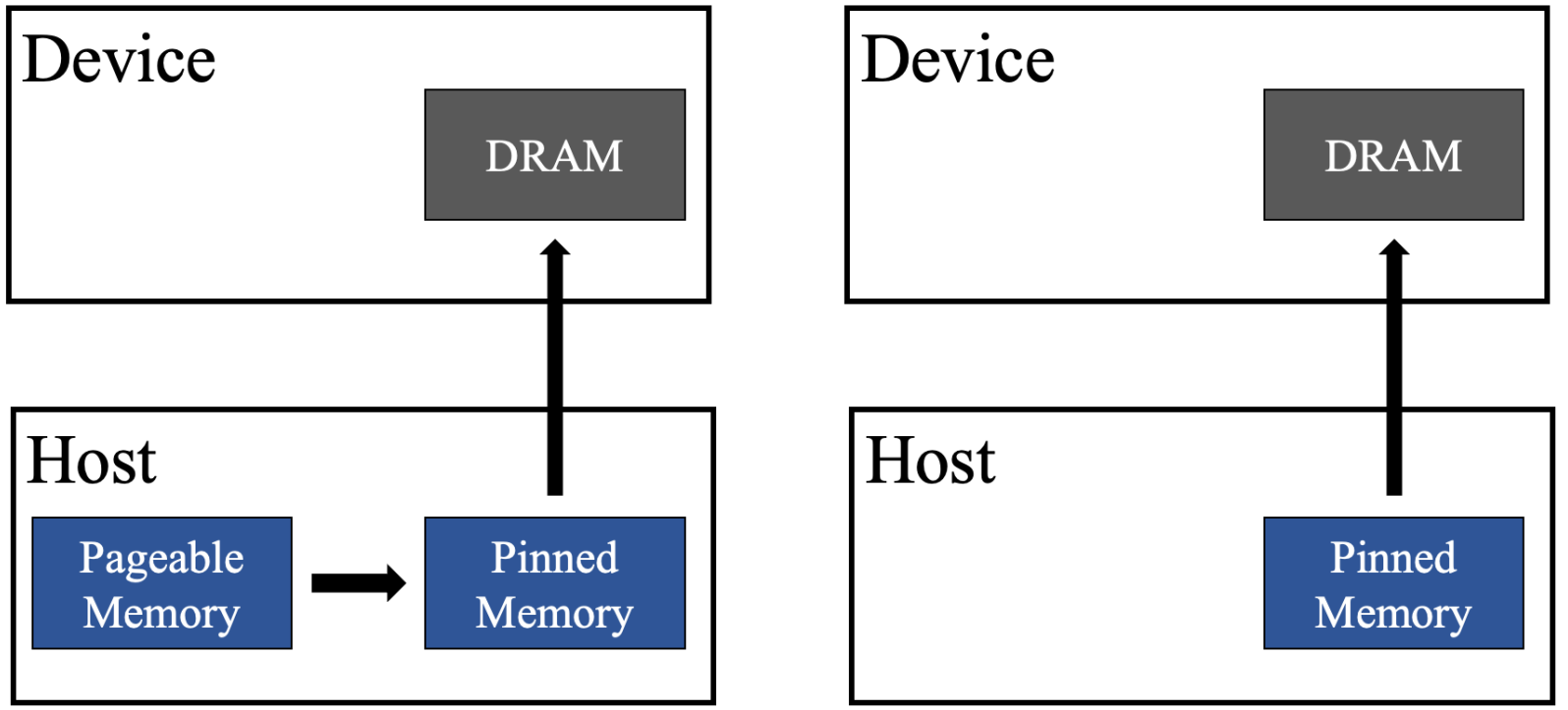
-03	밀리 초
기존 코드	0.196294
MixColumns 최적화 적용 후	0.204372
ShiftRows 최적화 적용 후	0.203700

- 그러나 둘다 적용하면 속도가 빨라진다.

-03	밀리 초
기존 코드	0.196294
둘다 적용 후	0.165780

4. GPU 에서의 구현

- Pinned 메모리 사용
  - Malloc() → cudaMallocHost() or cudaHostAlloc() 사용



Pageable Data Transfer

Pinned Data Transfer

4. GPU 에서의 구현

- Pinned 메모리 사용

```
u8* pt = (u8*)malloc(sizeof(u8) * NUM * 16);
u8* ct = (u8*)malloc(sizeof(u8) * NUM * 16);
u8 roundkey[12 * 16];
```

```
cudaStatus = cudaHostAlloc((void**)&roundkey, sizeof(u8) * 12 * 16, cudaHostAllocDefault);
cudaStatus = cudaHostAlloc((void**)&pt, sizeof(u8) * NUM * 16, cudaHostAllocDefault);
cudaStatus = cudaHostAlloc((void**)&ct, sizeof(u8) * NUM * 16, cudaHostAllocDefault);
```

4. GPU 에서의 구현

- 테이블 확장에서 10개 Thread를 사용하여 병렬 연산

```
void Get_VSboxAll(pilsung_ctx * ctx) {
    for(size_t rounds = 1; rounds < g_round_cnt; ++rounds) {
        for(size_t i = 0; i < 4; ++i) {
            for(size_t j = 0; j < 4; ++j) {
                Get_P8forSEnc(ctx->e_key[j + 4 * i + 16 * rounds], ctx);
                for(size_t k = 0; k < 256; ++k) {
                    ctx->sboxes[rounds][i][j][k] = Get_PeSb(SubByte(k), ctx);
                }
            }
        }
    }
}


// Generate all necessary P-boxes
void Get_VPboxAll(pilsung_ctx * ctx) {
    for(size_t rounds = 1; rounds < g_round_cnt; ++rounds) {
        Get_P16Enc(&ctx->e_key[16 * rounds], ctx->pboxes[rounds]);
    }
}
```

```
__device__ void Get_VSboxAll(pilsung_ctx* ctx, u32 tid, u8* Tree_Integer8, u8* Tree_Integer4) {
    for (size_t i = 0; i < 4; ++i) {
        for (size_t j = 0; j < 4; ++j) {
            Get_P8forSEnc(rk[j + 4 * i + 16 * tid], ctx, tid, Tree_Integer8, Tree_Integer4);
            for (size_t k = 0; k < 256; k++) {
                ctx->sboxes[tid][i][j][k] = Get_PeSb(SubByte(k), tid, ctx);
            }
        }
    }
}


// Generate all necessary P-boxes
__device__ void Get_VPboxAll(pilsung_ctx* ctx, u32 tid, u8* Tree_Integer4) {
    Get_P16Enc(&rk[16 * tid], ctx->pboxes[tid], Tree_Integer4);
}
```

```
if (tid > 0 && tid < 11) {
    gen_enc_perm(ctx, tid, Tree_Integer8, Tree_Integer4);
}
__syncthreads();

pilsung_encrypt(ctx, out, in);
```

Impl.	Thread					
	32	64	128	256	512	1024
None	10	14.9	24.1	40.7	75.3	145.2
O	9.6	12.7	20.3	35.9	69.3	142.1
P	8.9	12.7	19.5	32.1	58.4	112.7
PO	8.4	10.7	15.6	27.3	51.8	108.5

감 사 합 니 다