FF1 and CTR mode of AVX2-PIPO

https://youtu.be/XaDw6lXSpNc

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

Contents

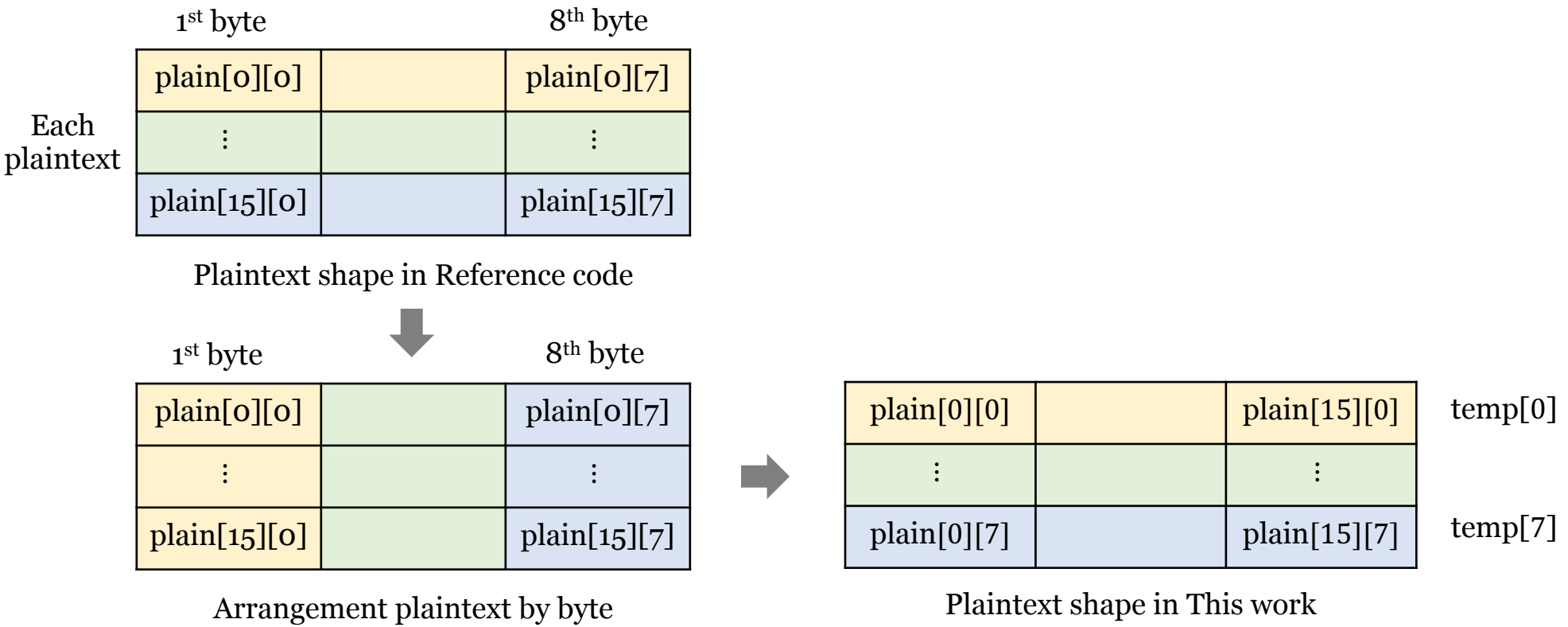
CryptoCraft LAB

AVX2-PIPO 구현 정리

- Plaintext 설정

  16개의 평문 동시에 연산하도록 256-bit 레지스터에 바이트 단위로 모음

  최대 32개 가능하지만 효율적인 rotation 위해 변경

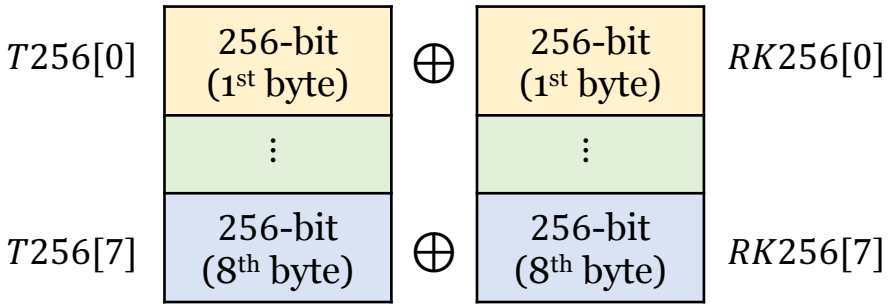


AVX2-PIPO 구현 정리

- Key Add

  라운드키 (모든 평문에 동일)을 1byte 단위로 256-BIT 레지스터에 저장하여

  동일한 위치의 바이트에 대해 평문 256-bit와 XOR



AVX2-PIPO 구현 정리

- S-Layer

  256-bit 레지스터를 사용 → 기존 PIPO와 동일하게 접근 가능 (인덱스..)

  그러나 AVX2는 NOT 연산을 지원하지 않아서 ANDNOT을 통해 구현

AVX2-PIPO 구현 정리

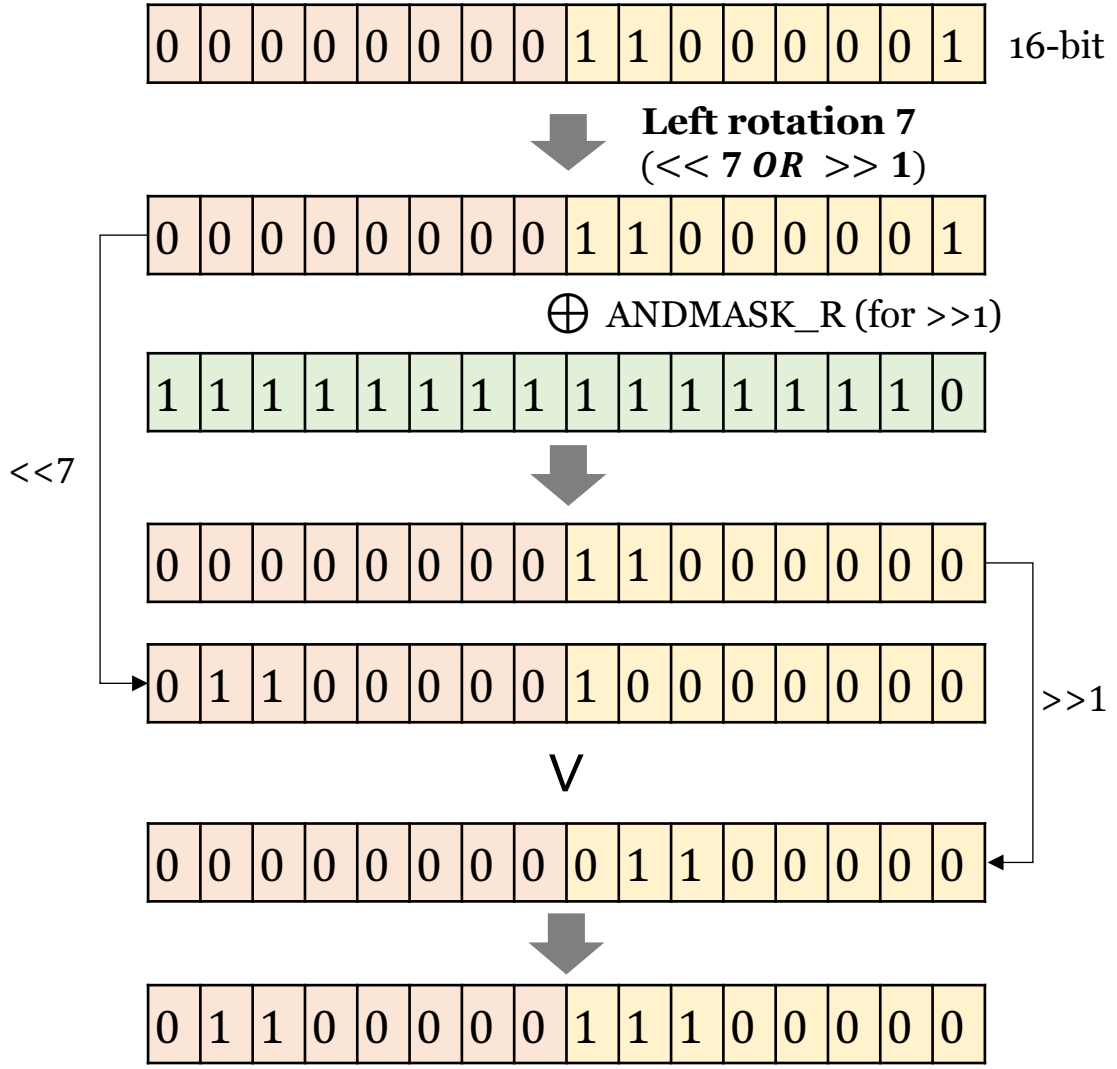
- R-Layer

  16-bit rotation 구현

  → 지난 번에는 8-bit를 16-bit로 바꿔서 했는데

  처음부터 16bit 평문 16개로 해서

  8-bit to 16-bit 변환 과정을 없앰 → 성능 향상



AVX2-PIPO 정리

- 구현 결과

```
ROUND 13: 6B6B2981, AD5D0327
```

```
6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b
6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81
ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad
5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27
```

```
6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b
6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b 6b
29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29 29
81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81 81
ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad ad
5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d 5d
03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03 03
27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27
```

그림 12. 기존 PIPO(위)와 AVX2-PIPO(PT 16, PT 32)의 마지막 라운드 출력값 비교

AVX2-PIPO 성능 평가

- 성능 비교

표 3. Comparison Result Table (Speed) : Reference Code, AVX2-PIPO32 and AVX2-PIPO16 (-O1, unit : ms)

The number of PT	Ref. C	AVX2-PIPO32	AVX2-PIPO16
32	0.501	0.505	0.502
64	0.511	0.509	0.503
320000	54.647	32.876	9.003
3200000	451.136	260.624	73.538
32000000	4390.659	2437.528	597.735

3200만개의 평문 블록에 대해
Ref.C에 비해
AVX2-PIPO32 : 1.801배
AVX2-PIPO16 : 7.345배의 연산 속도

표 4. Comparison Result Table (cpb): Reference Code, AVX2-PIPO32 and AVX2-PIPO16 (-O1, unit : cpb)

Ref. C	AVX2-PIPO32	AVX2-PIPO16
44.592	24.75	6.07

AVX2-PIPO16의 성능이 가장 좋으며,
Ref. C에 비해 약 7.34배의 성능향상

AVX2-PIPO-FF1

- FF1의 roundfunction에는 PRF와 CIPH 함수 존재
- PRF, CIPH 에는 AES128을 사용
- PRF

  길이가 16byte로 고정된 값을 초기 블록으로 입력

  즉, 입력 평문이 길어도 PRF는 128bit 하나를 입력 받음

  병렬 처리없이 AES-128-CBC를 사용
- CIPH

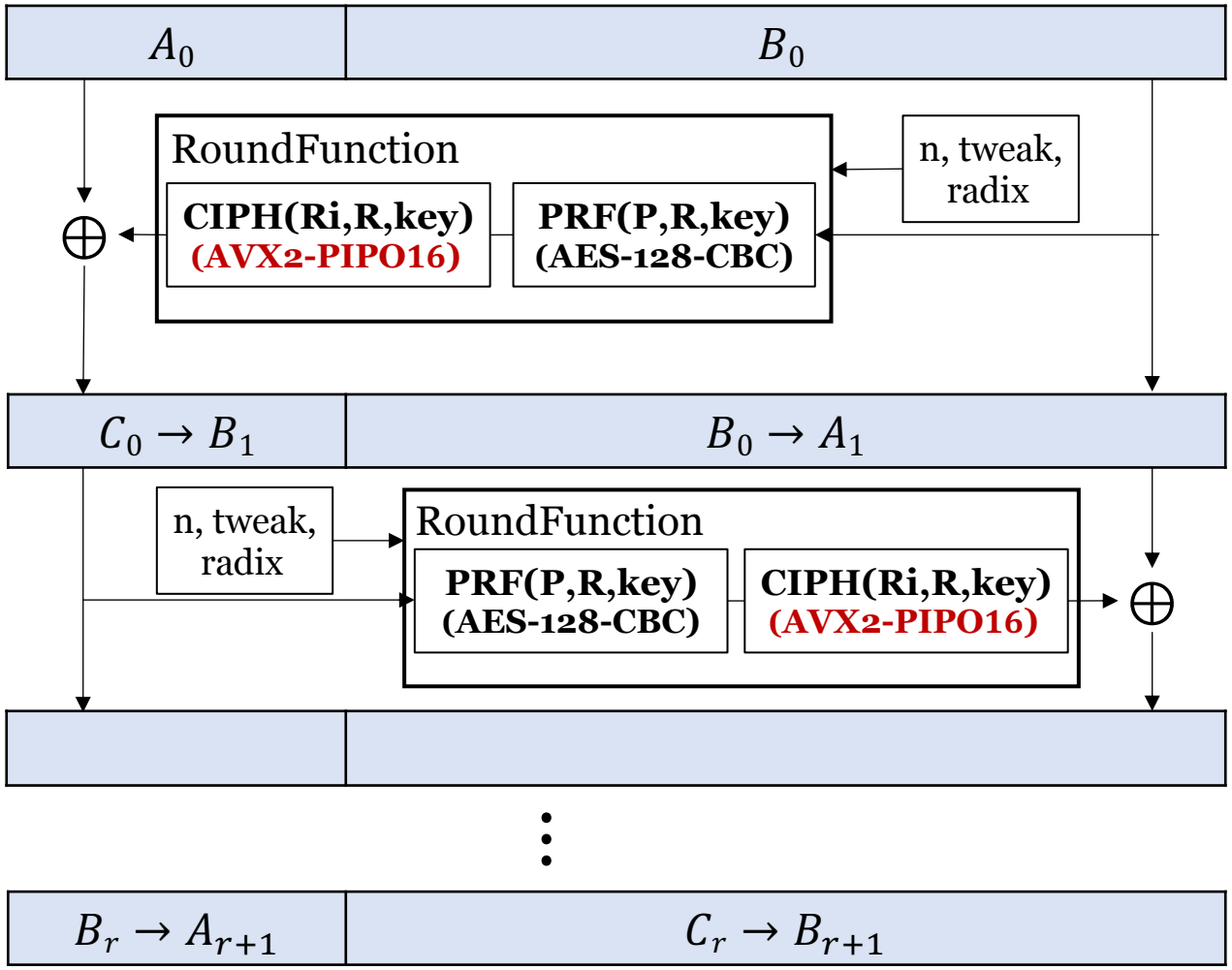
  Q (tweak, d 등에 의해 길이 결정)와

  PRF의 output (16byte)을 입력

  → 기존 FF1의 경우, 암호화 한 번 수행 후

  Q 16바이트 이동 후 Ri와 다시 XOR 후 암호화 …

  → 제안 방법의 경우, 16개 평문에 대해 한 번에 암호화



AVX2-PIPO-FF1 구현 결과

- 평문 길이가 유지되면서 암호화

```
ption-master$ ./example 2B7E151628AED2A6ABF7158809CF4F3C 39383736353433323130 10
 0123456789
key: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
tweak: 39 38 37 36 35 34 33 32 31 30
ret : 0
after map:  0 1 2 3 4 5 6 7 8 9

========== FF1 ==========
ciphertext: 9114946704

plaintext: 0 1 2 3 4 5 6 7 8 9
```

그림 13-(a). 평문 길이 10, 암호문 길이 10, radix 10

```
ption-master$ ./example 2B7E151628AED2A6ABF7158809CF4F3C 39383736353433323130 30
 0123456789abcdefgh
key: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
tweak: 39 38 37 36 35 34 33 32 31 30
ret : 0
after map:  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

========== FF1 ==========
ciphertext: a24lrjttdjbe16t98m

plaintext: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
```

그림 13-(b). 평문 길이 18, 암호문 길이 18, radix 30

```
ption-master$ ./example 2B7E151628AED2A6ABF7158809CF4F3C 39383736353433323130 30
 0123456789abcdefghk12397421651213
key: 2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c
tweak: 39 38 37 36 35 34 33 32 31 30
ret : 0
after map:  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 20 1 2 3 9 7 4 2 1 6 5 1
 2 1 3

========== FF1 ==========
ciphertext: 6sr66jpfb2jpkb805ohq55nqc4ofj8toi

plaintext: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 20 1 2 3 9 7 4 2 1 6 5 1
2 1 3
```

그림 13-(c). 평문 길이 33, 암호문 길이 33, radix 30

AVX2-PIPO-FF1 응용 및 성능 비교

- 데이터 베이스 암호화에 활용

  → MySQL이 제공하는 AES-128, SHA2, DES 함수 대신 구현한 AVX2-PIPO-FF1 사용

- MySQL Query 사용해서 AVX2-PIPO-FF1 소스코드 내에서 작성 가능

```
MYSQL *conn = mysql_init(NULL);
Set_Connection(conn); //mysql 연결
//Create_DB(conn); //DB생성
//Create_Table(conn); //Table생성


mysql_query(conn, "USE PIPO_FF1");
// 여기서 DB 암호화를 할 것임..
Insert_EncData(x, y, xlen, &ff1, FPE_ENCRYPT, k);
```

```
mysql> SELECT * FROM FF1_PIPO;
+------+------------------------------------+
| ID   | PW                                 |
+------+------------------------------------+
| id05 | 9114946704                         |
| id05 | a24lrjttdjbe16t98m                 |
| id05 | 6sr66jpfb2jpkb805ohq55nqc4ofj8toi  |
+------+------------------------------------+
3 rows in set (0.01 sec)
```

그림 14. AVX2-PIPO16-FF1으로 암호화된 데이터가 저장된 데이터베이스

```
void Insert_EncData(unsigned int *in, unsigned int *out, unsigned int inlen, FPE_KEY *key, const int enc ,const unsigned char *userKey)    {

    if (enc)
        FF1_encrypt_PIPO(in, out, &key->aes_enc_ctx, userKey, key->tweak, key->radix, inlen, key->tweaklen);

    else
       FF1_decrypt_PIPO(in, out, &key->aes_enc_ctx,userKey, key->tweak,key->radix, inlen, key->tweaklen);
```

AVX2-PIPO-FF1 응용 및 성능 비교

- MySQL 환경 문제로 인해 2GB RAM Ubuntu 16.04 LTS 가상머신 상에서 수행

표 5. Comparison Result Table (Speed) : AES, SHA2 of MySQL and AVX2-PIPO16
(unit : ms)

Data	AES	SHA2	AVX2-PIPO16-FF1
A1234	0.755	0.818	0.592
01234567890123456789012345678901234567890123456789ABCDEFGH	0.734	0.874	0.852

AVX2-PIPO-FF1 응용 및 성능 비교

- 사용한 암호알고리즘별로 데이터베이스 및 테이블 생성 후 메모리 사용량 비교
- 짧은 길이의 평문 (A1234), 긴 평문 (01234567890123456789012345678901234567890123456789012345678901234567890123456789ABCDEFGH)을 각각 10만번씩 3번 저장한 후의 메모리 사용량
  (각 DB당 총 60만 개의 데이터)
- SHA2 > AES > AVX2-PIPO-FF1 순으로 많은 메모리 공간 사용
  → SHA2 (256-bit), AES (128-bit), AVX2-PIPO-FF1 (평문과 동일)
  FF1의 특성상 평문과 암호문의 길이가 동일하여 패딩 필요 x
  → 메모리 공간 낭비 x
- 형태도 유지 : 추가적인 데이터베이스 스키마 변경이 필요하지 않음
  따라서 데이터베이스 관리 비용 증가 및 시스템 수정이 거의 발생 x

DB Name	MB
AES	69.6
information_schema	0.2
mysql	2.5
performance_schema	0.0
PIPO_FF1	38.6
SHA	98.6
sys	0.0

그림 15. 각 데이터베이스 메모리 사용량

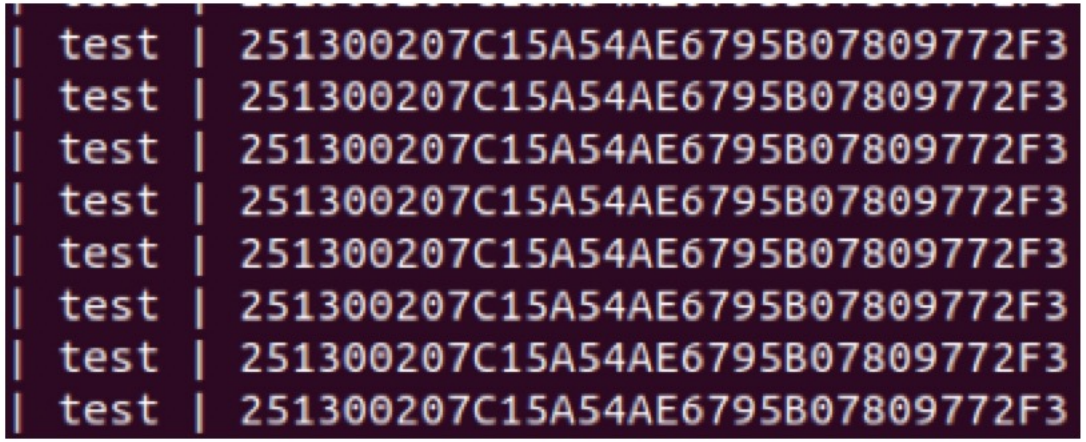


그림 16-(a) AES를 사용하여 암호화 된 후 저장된 데이터 (128-bit 고정)

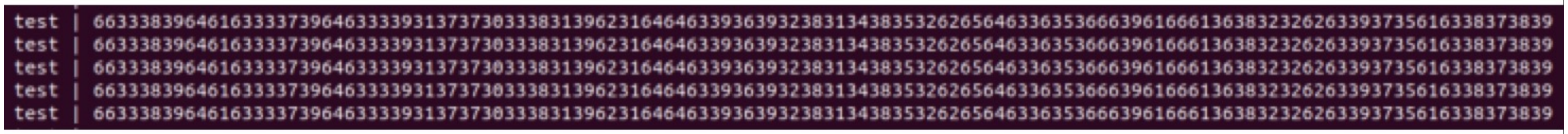


그림 16-(b) SHA2를 사용하여 암호화 된 후 저장된 데이터 (256-bit 고정)

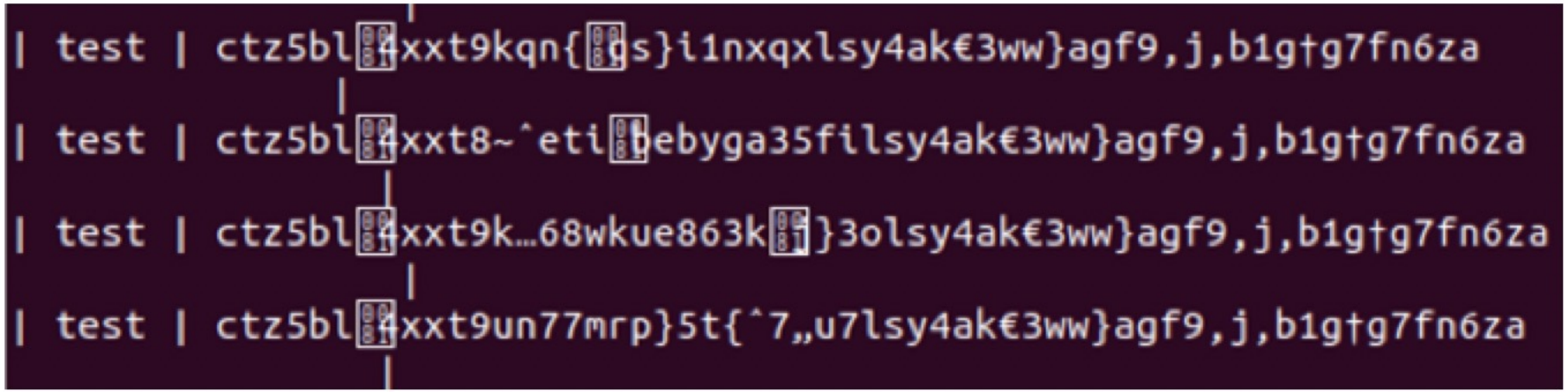
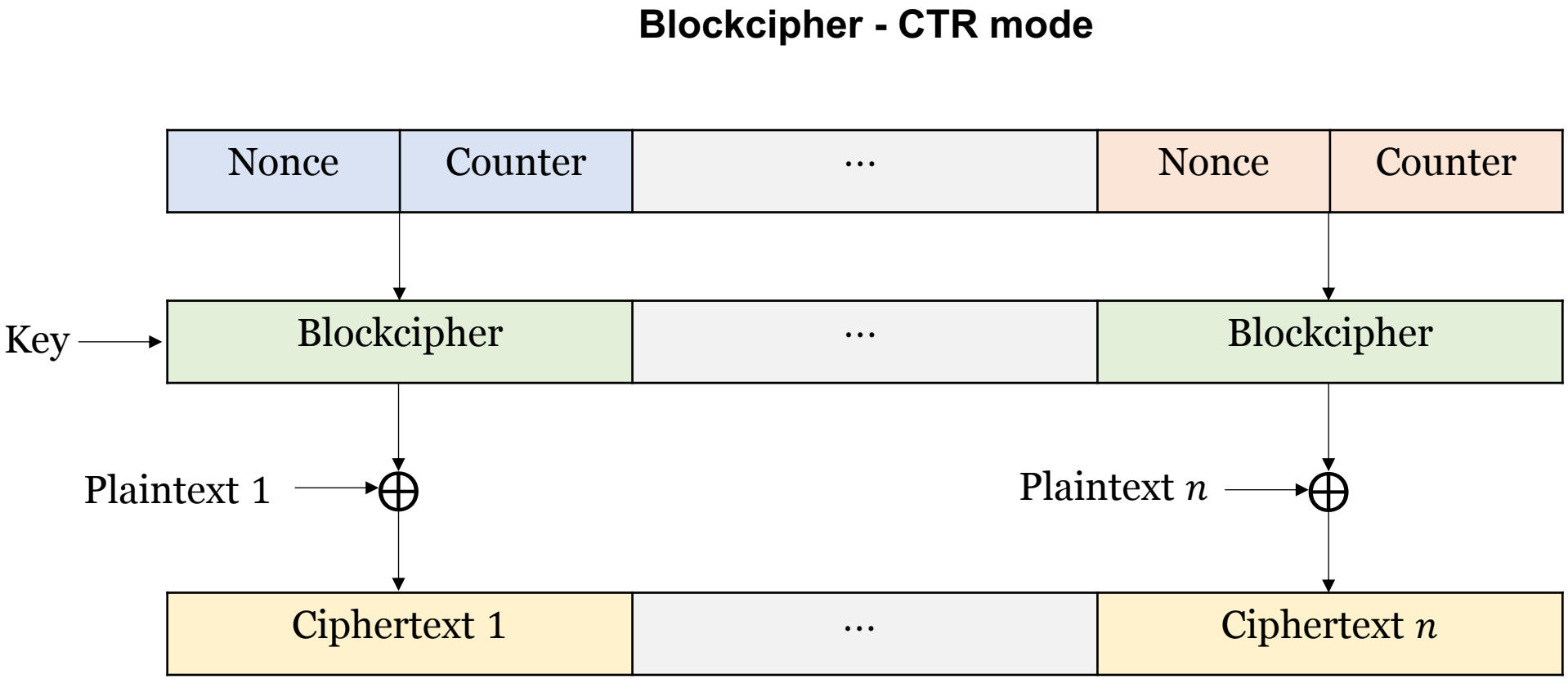


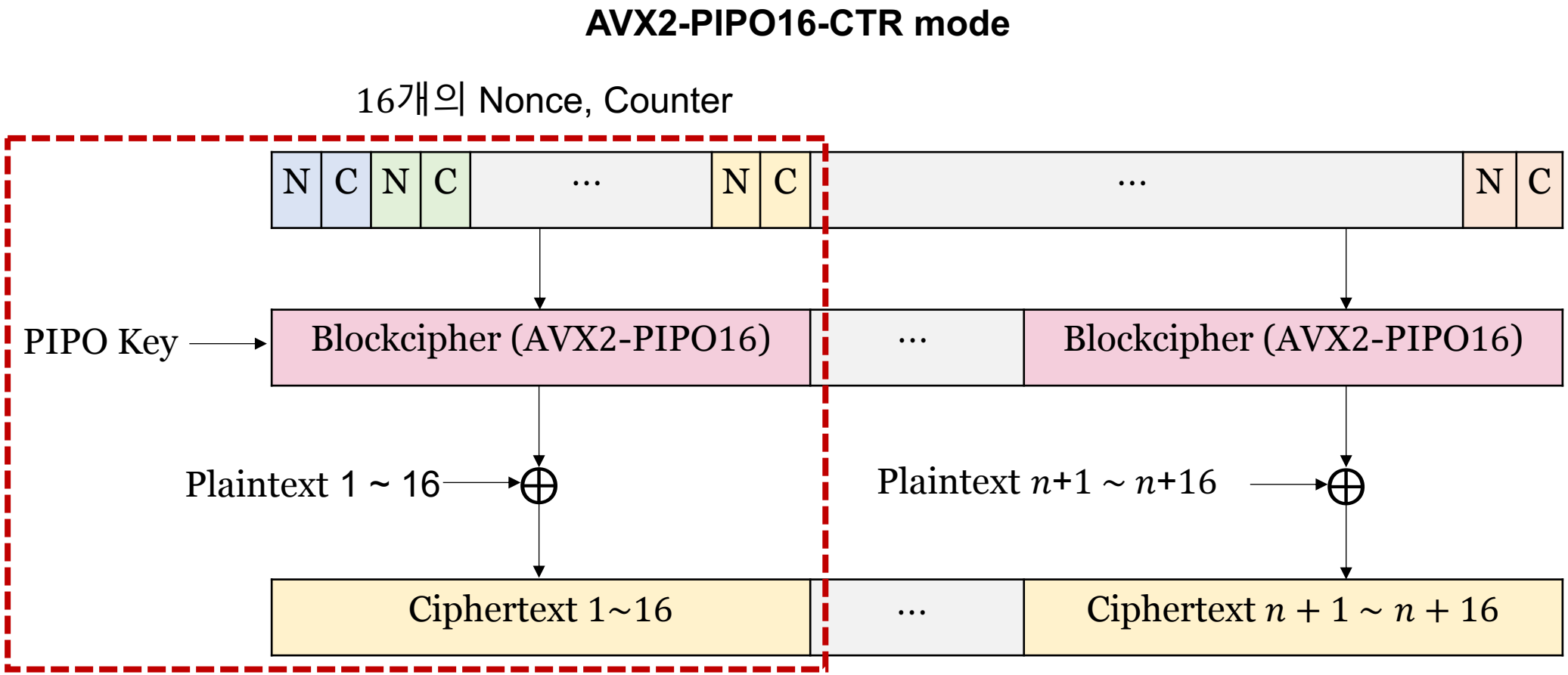
그림 16-(c) AVX2-PIPO16-FF1을 사용하여 암호화 된 후 저장된 데이터 (입출력 길이 보존)

그림 16 MySQL의 AES, SHA2 그리고 AVX2-PIPO-FF1의 암호화 결과

AVX2-PIPO-CTR



AVX2-PIPO-CTR



AVX2-PIPO-CTR 구현 결과

- 왼 : nonce 0x01,02,03,04로 고정 + 2번째 평문에 카운터++한 값으로 지정하여 일반 PIPO로 돌린 결과
- 오 : nonce 값 동일하게 고정 (확인용)후 CTR mode 적용 (왼쪽과 2번째 블록까지 동일하면 됨..)

```
[kimhyunji@gimhyeonjiui-MacBookPro 16test % ./tt
6b 54 6b 6b 6b 6b 6b 6b
70 e0 70 70 70 70 70 70
9c e9 9c 9c 9c 9c 9c 9c
ba b1 ba ba ba ba ba ba
f5 e9 f5 f5 f5 f5 f5 f5
d8 d5 d8 d8 d8 d8 d8 d8
6c 62 6c 6c 6c 6c 6c 6c
77 60 77 77 77 77 77 77
```

```
for (int i=0; i<16;i++){    //test vector
        plain[i][0]= 0x01;
        plain[i][1]= 0x02;
        plain[i][2]= 0x03;
        plain[i][3]= 0x04;
        plain[i][4]= 0x00;
        plain[i][5]= 0x00;
        plain[i][6]= 0x00;
        plain[i][7]= 0x00;
}

plain[1][0]= 0x01;
plain[1][1]= 0x02;
plain[1][2]= 0x03;
plain[1][3]= 0x04;
plain[1][4]= 0x00;
plain[1][5]= 0x00;
plain[1][6]= 0x00;
plain[1][7]= 0x01;
```

```
[kimhyunji@gimhyeonjiui-MacBookPro AVX2-PIPO16-CTR % ./test
6b 54 a9 c3 47 13 bb eb ab 68 c7 e7 4a 22 48 ea
70 e0 86 8b ca 57 89 5c 68 88 01 69 d9 d8 e7 7f
9c e9 b3 3d e4 45 9d 54 23 4a 1b f6 0d b9 34 44
ba b1 ac 0c e9 36 43 e1 7c e4 ca 2f db e8 a0 11
f5 e9 bb c3 30 8a 9b cf 41 be fd 5b a9 c6 30 2b
d8 d5 85 2e 9c 1d 29 4e 52 e7 de 02 77 b1 4e 21
6c 62 20 13 6f 76 1f bb 16 8e 69 ed ab 77 12 72
77 60 13 67 a9 51 3b 92 52 4f 02 72 7b 68 c0 9e
```

AVX2-PIPO-CTR 구현 결과

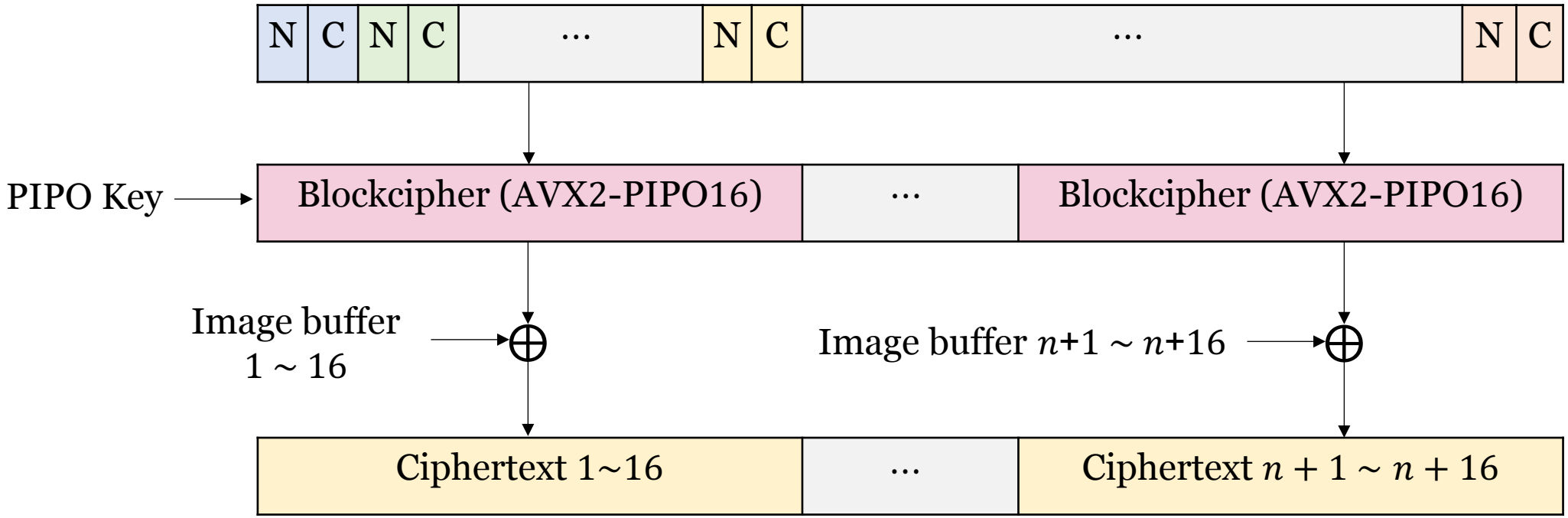
- 왼쪽 위 : PIPO-CTR 돌린 후 (키스트림)
- 왼쪽 아래 : 위 XOR 평문
- 위 XOR 아래 → 평문

```
eb 04 27 97 76 3d b1 ce 48 a7 f4 26 05 d6 94 9a
ab 04 19 11 0f 36 b7 fa 37 c2 c6 9c 6f 10 d3 5b
5b b9 bd c1 7a 51 0f 54 5b bd ab bb 8c 06 4b f6
91 ef ac 7a 8a de 1b 31 63 b3 fe 23 31 79 3e b9
bf a7 8e e1 79 c2 ba bf ad c2 66 08 b3 47 a7 9a
a2 bb 92 83 66 8d ac 4b 90 25 cc 09 9e 2f df 70
3a 77 28 b6 ed 05 ac e1 7f 86 0d ff 5d 41 6a 46
22 ba 45 12 64 72 2c c0 9d 4c 79 93 bd 6a 58 b7
======= After XOR =======
e2 0d 2e 9e 7f 34 b8 c7 41 ae fd 2f 0c df 9d 93
2e 81 9c 94 8a b3 32 7f b2 47 43 19 ea 95 56 de
09 eb ef 93 28 03 5d 06 09 ef f9 e9 de 54 19 a4
67 19 5a 8c 7c 28 ed c7 95 45 08 d5 c7 8f c8 4f
a1 b9 90 ff 67 dc a4 a1 b3 dc 78 16 ad 59 b9 84
85 9c b5 a4 41 aa 8b 6c b7 02 eb 2e b9 08 f8 57
3a 77 28 b6 ed 05 ac e1 7f 86 0d ff 5d 41 6a 46
04 9c 63 34 42 54 0a e6 bb 6a 5f b5 9b 4c 7e 91
```

```
for (int i=0; i<16;i++){    //test vector
        plain[i][0]= 0x26;
        plain[i][1]= 0x00;
        plain[i][2]= 0x27;
        plain[i][3]= 0x1E;
        plain[i][4]= 0xF6;
        plain[i][5]= 0x52;
        plain[i][6]= 0x85;
        plain[i][7]= 0x09;
}
```

AVX2-PIPO-CTR 응용

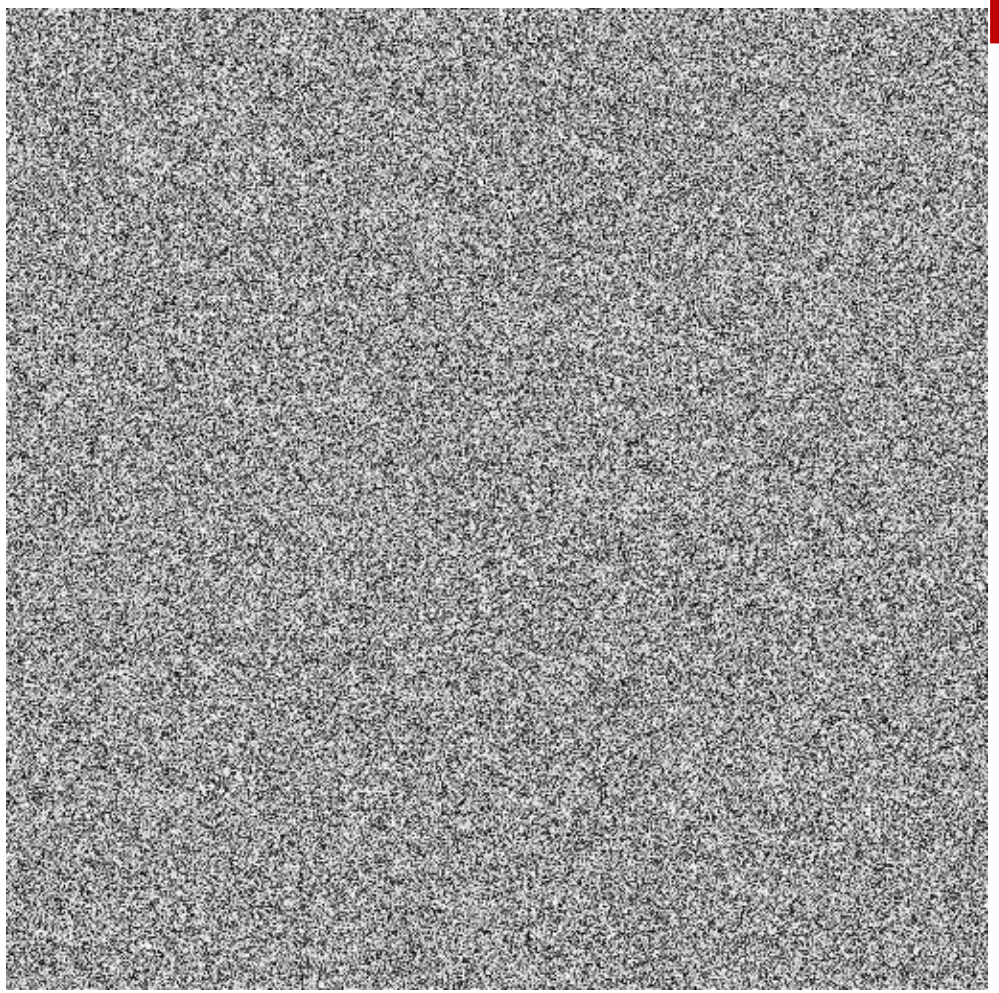
- Image를 오픈하여 buffer에 128-byte씩 읽어와서 암호화한 후 저장
- Image buffer는 plaintext에 해당하므로 미리 생성된 키 스트림 (AVX2-PIPO-CTR(ctx,roundkey))과 XOR 되기만 하면 됨



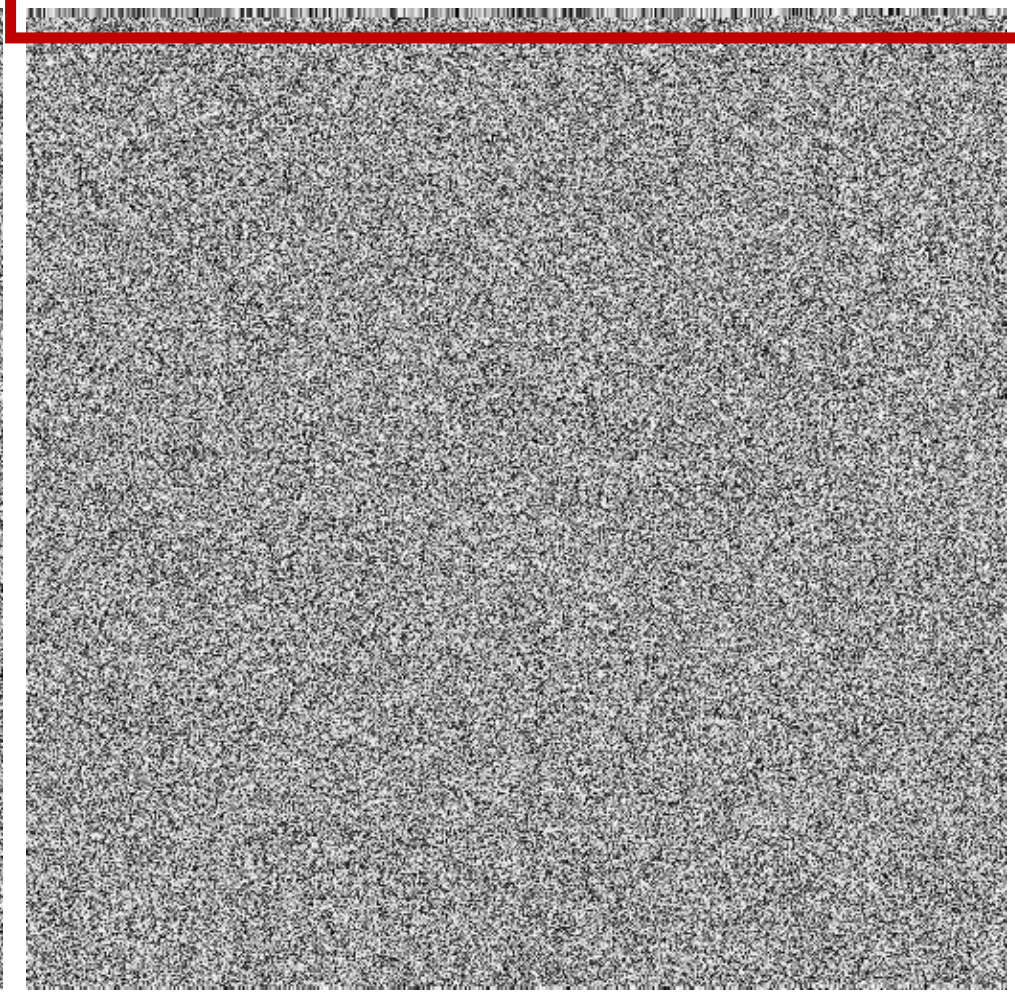
AVX2-PIPO-CTR 응용

- 암호화 결과

원본 512x512



CTR mode



그냥

향후 계획

- 이미지 암호화

  1. 동영상으로 변경 ..

  2. 빨간 박스 부분 왜 그런건지.. 잘못된건지..확인..

- 객체 탐지 후 부분 암호화 (코드는 작성했는데 가상머신 고장나서 새로 깔고 해보겠습니다)
- GCM 모드도 해보려고 했는데 시간 되면 해보겠습니다
- PIPO 공모전에서 쿠다 관련 부분은 어떻게 해야할지 여쭤보고싶습니다..

Q & A