2024 전기 졸업과제 중간 보고서

RTOS환경 취약점 분석 및 경량화

지도교수: 손준영

202155654임준식

202155656정혁준

202155644김성문

팀명: 운영체제분석

분과: 하드웨어/보안(D)

목차

1. 요구조건 및 제약 사항 분석에 대한 수정사항

1.1 기존의 요구 조건

- 현재 존재하는 알고리즘에 대한 분석을 진행하고 경량화를 진행.
- QNX의 기존에 존재하는 취약점을 분석.
- QNX를 이용하여 원래 없던 취약점을 발견.
- 취약점으로 인해서 발생할 수 있는 문제점을 찾기.

1.2 요구 조건에 대한 수정사항

- QNX를 이용하여 발견하지 못했던 취약점을 발견하는 건 학부생의 수준에서는 어렵다고 판단을 하였습니다. 그래서 QNX에 기존에 존재하는 취약점을 QNX를 대상으로 실행해보는 걸 목표로 취약점을 분석하고 실제로 실행해보는 방향으로 졸업 과제를 진행하기로 했습니다.

1.3 제약 사항 분석

- IAR툴에 대한 라이선스는 교육용 라이선스를 이용해서 해결을 하였다. 또한 툴을 사용하는 방법을 학교에서 세미나 진행을 통해 익혀서 졸업과제를 진행하면서 막히는 부분을 해결하였다.
- QNX툴에 대한 라이센스 권한을 교육용으로 받았으나 비밀번호와 관련된 문제가 있어서 QNX본사에 해결방안을 요청하였습니다.
- 보안과 관련해서 생소하다고 느끼는 부분은 각자 공부해서 알려주는 방식으로 보완을 하였습니다.
- IAR툴을 이용해서 경량화를 진행하였습니다. 하지만 툴이 임베디드 시스템을 개발할때 사용하는 툴이기 때문에 부가적으로 필요한 부품들이 고가의 제품들이 많이 있어서 CLion, visual studio를 이용해서 작성을 하고 IAR로 확인할 수 있는 Cycle수, register사용부분, 어셈블리어를 확인해서 경량화를 진행하였습니다.

2. 갱신된 과제 추진 계획

제약 사항 분석 부분에서 QNX에 아직 발견하지 못한 취약점을 발견하는 건 학부생 수준에서는 어렵다고 판단을 했기 때문에. IAR을 이용한 알고리즘 경량화는 계속 진행하지만, RTOS환경 취약점 분석은 기존의 취약점을 이용해서 분석을 진행하기로 하였습니다.

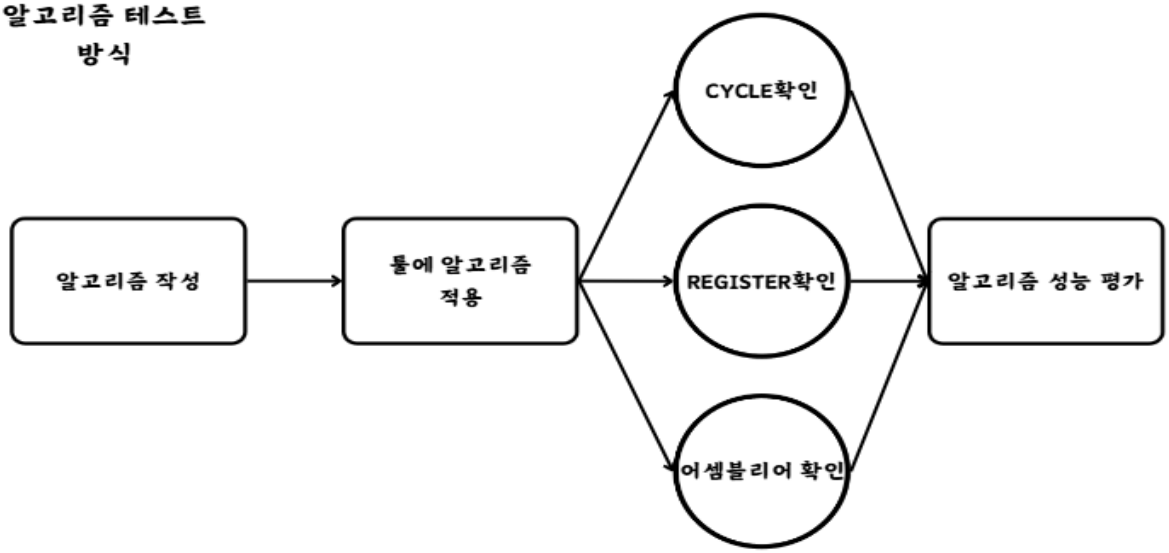
3. 구성원별 진척도

이름	내용
임준식	1. AES알고리즘 분석 및 경량화 진행. 2. ECB, CBC 운영 모드 조사 3. 피드백에 대한 보완점 조사
김성문	1. ARIA알고리즘 분석 및 경량화 진행. 2. CFB, OFB 운영 모드 조사 3. QNX취약점 분석
정혁준	1. ECC알고리즘 분석 및 경량화 진행. 2. CTR 운영 모드 조사 3. QNX취약점 분석
공통	1. IAR, QNX에 적용된 보안 알고리즘 조사 2. IAR, QNX의 보안 이슈사항 3. 운영 모드 적용(ECB, CBC, CFB, OFB, CTR, GCM) 4. 보안 알고리즘 조사 5. 보안에 관한 지식 학습

현재까지는 IAR툴을 이용해서 암호 알고리즘을 경량화 하는데 초점을 맞춰서 과제를 진행하였습니다. 9월 ~ 11월은 알고리즘의 경량화는 지속적으로 수행하면서 QNX툴을 이용한 취약점 분석에 초점을 맞춰서 기존의 취약점을 실제로 적용해보는 방향으로 졸업과제를 수행할 것입니다.

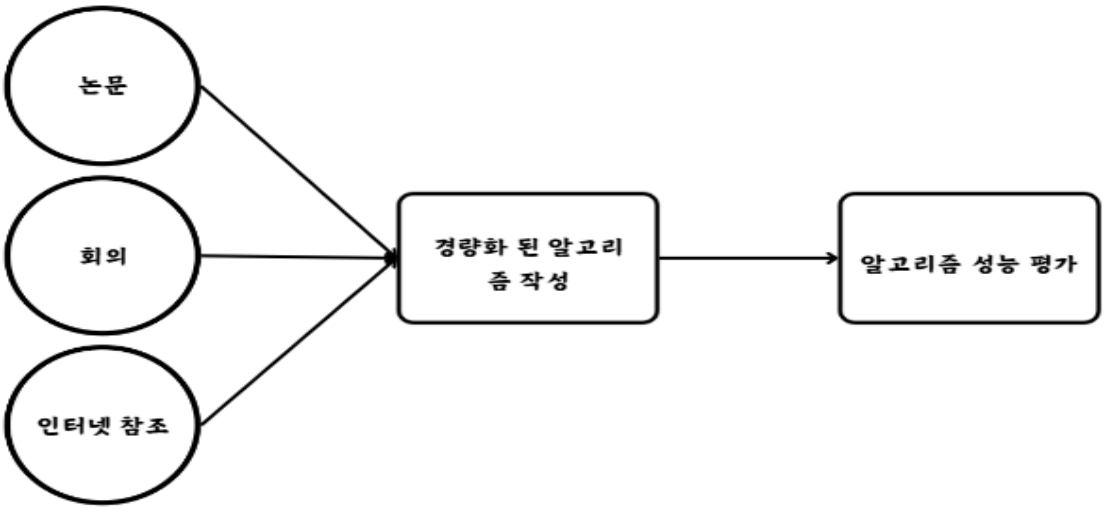
4. 설계 상세화 및 변경 내역

알고리즘 테스트 방식



다음과 같은 방식으로 알고리즘을 작성하고 난 이후에 성능을 평가하는 과정을 수행했습니다. 시간의 경우 모두가 동일한 환경의 임베디드 제품을 활용해서 진행한 게 아니기 때문에 시간을 제외하고 CYCLE을 확인하는 방법으로 알고리즘의 성능을 평가하였습니다. 여기서 사용한 툴은 **IARworbench**를 이용해서 성능을 평가하였습니다.

알고리즘 최적화 방식

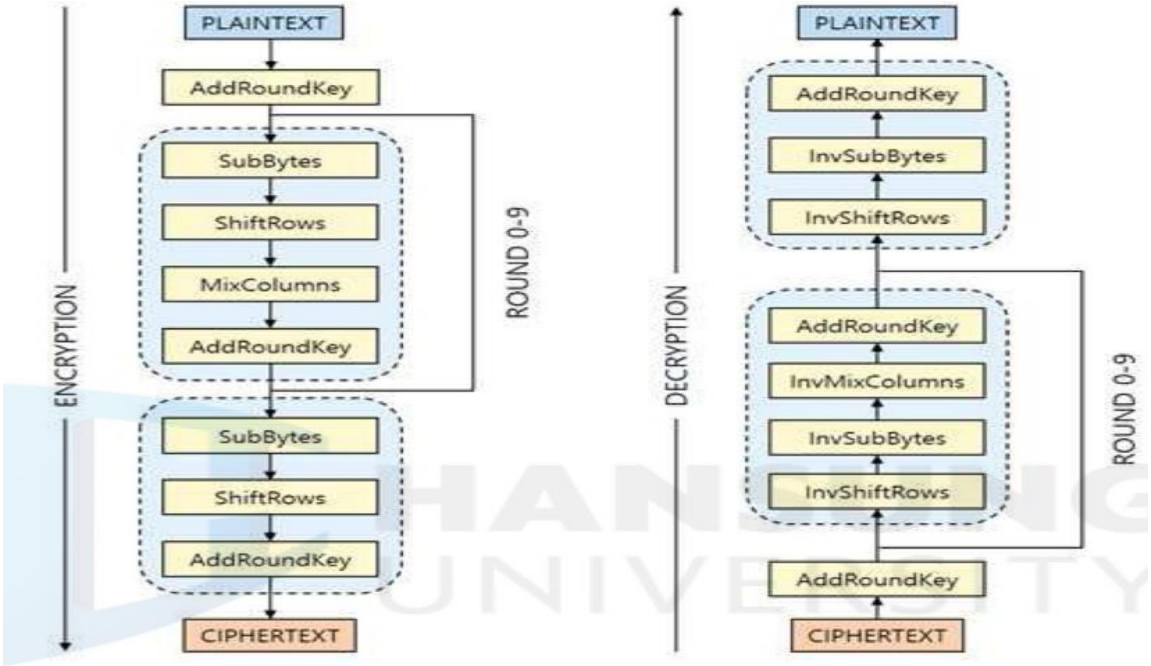


알고리즘 테스트를 하고 난 이후에는 성능을 최적화하기 위해서 논문, 회으, 인터넷 참조를 이용해서 경량화된 알고리즘을 만들었습니다.

5. 보고 시점까지의 과제 수행 내용 및 중간결과

5.1. 보안알고리즘 조사

5.1.1 AES



[그림 2-1] AES 암호 알고리즘

대칭키 암호 알고리즘으로 고정된 128-bit블록 크기를 가진다. 키의 길이에 따라서 128/192/256-bit로 나눠지고 각각 10/12/14회의 라운드를 거치며 진행된다.

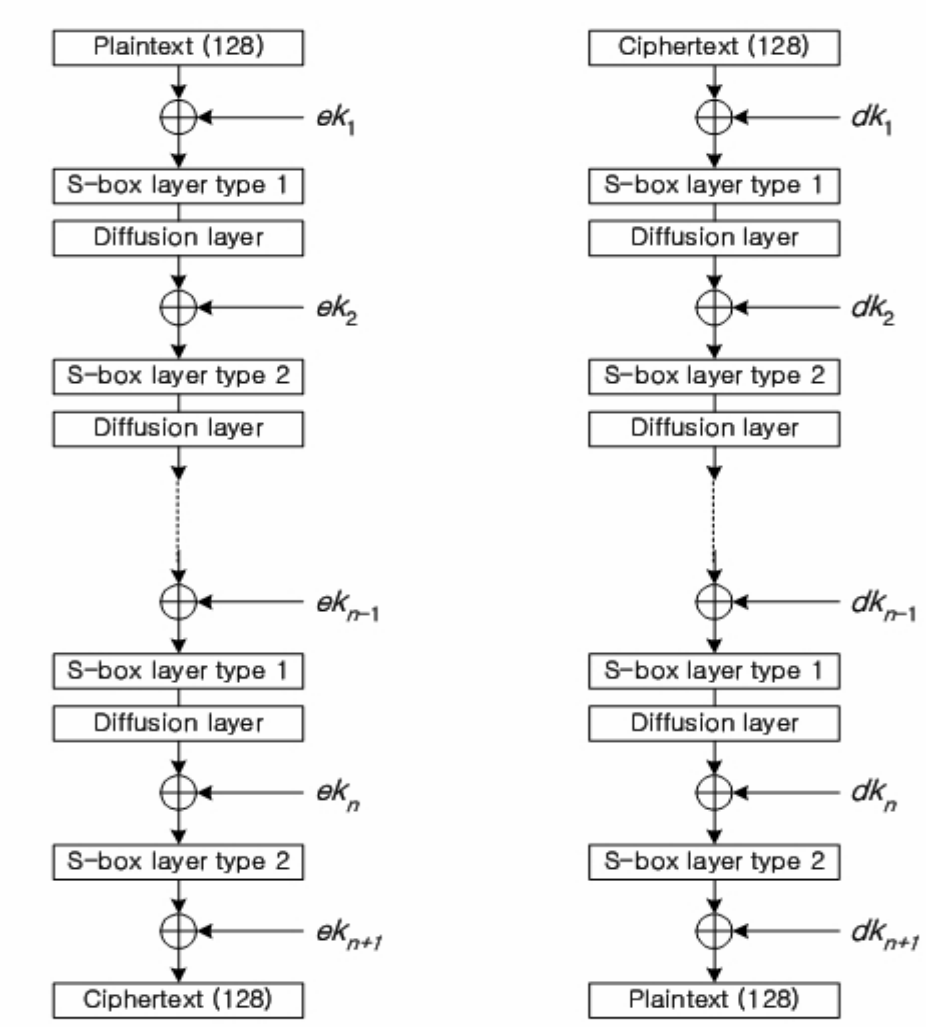
AddRoundKey: XOR연산을 수행하고 연산당 1개의 바이트에만 영향을 준다.

SubBytes: 미리 저장된 S-BOX테이블을 활용하여 치환연산을 진행한다.

ShiftRows: 각 행의 순서를 바꾸는 연산.

MixColumns: 각 열의 값들을 곱셈 연산을 통해 암호화의 확산을 담당하며 AES암호 알고리즘의 연산 중 가장 긴 소요시간이 필요한 연산이다.

5.1.2 ARIA



출처 : ARIA 블록 암호 알고리즘 구현 - 한재수

ARIA 알고리즘은 대칭키 암호 알고리즘으로, AES 알고리즘과 마찬가지로 고정된 128-bit 블록 크기를 가진다. 키 길이에 따라 12(128bits)/14(192bits)/16(256bits)회의 라운드를 거치며 진행된다.

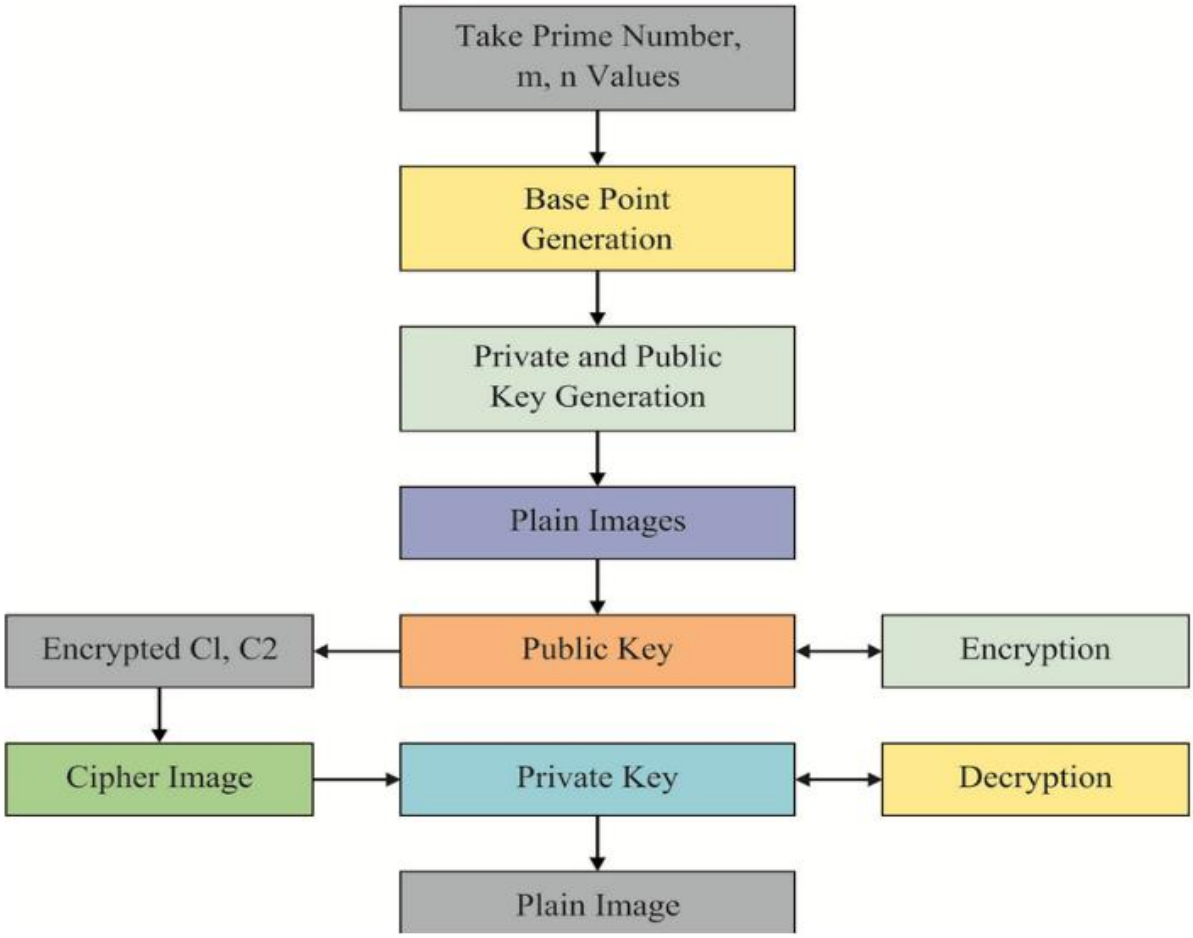
ek : 각 라운드의 키로, 이전 라운드의 출력(첫번째 라운드의 경우 평문)과 XOR된다.

s-box layer : 데이터를 비선형적으로 변환하는 layer로, 미리 정의된 값으로 구성된 substitution box를 활용해 입력된 데이터를 치환한다.

ARIA 알고리즘은 치환 계층에서의 보안성을 강화하기 위해 두 종류의 s-box를 교차로 사용하는데, 홀수 라운드에서는 S1이 사용되고, 짝수 라운드에서는 S2가 사용된다.

diffusion layer : 입력 데이터를 특정 규칙에 따라 섞어주는 layer로, 행렬 곱셈 연산을 통해 입력 데이터를 복잡하게 섞어준다.

5.1.3 ECC



출처 : ResearchGate

위 다이어그램은 ECC(타원 곡선 암호화, Elliptic Curve Cryptography) 알고리즘의 일반적인 흐름을 나타내고 있다.

다이어그램의 각 단계는 다음과 같다:

1. 소수 및 값 선택: 소수(prime number)와 두 개의 값(m, n)을 선택한다.

2. 기본 점 생성: 선택한 소수를 기반으로 기본 점(base point)을 생성한다.

3. 개인 키 및 공개 키 생성: 기본 점을 사용하여 개인 키(private key)와 공개 키(public key)를 생성한다.

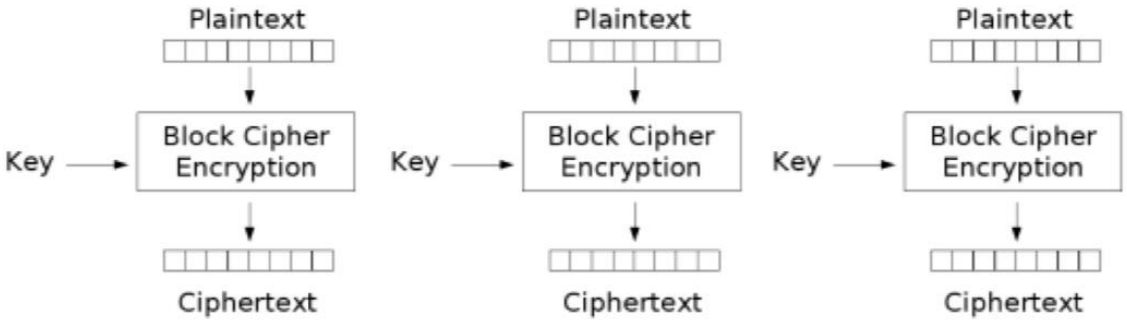
4. 원본 이미지: 암호화할 원본 이미지(plain images)를 준비한다.

5. 암호화: 공개 키를 사용하여 원본 이미지를 암호화(encryption)한다. 이 과정에서 암호화된 결과물(encrypted C1, C2)과 함께 암호화된 이미지(cipher image)가 생성된다.

6. 복호화: 개인 키를 사용하여 암호화된 이미지를 복호화(decryption)한다. 이 과정을 통해 원본 이미지(plain image)를 다시 얻는다.

5.2. 운영모드 조사

5.2.1 ECB

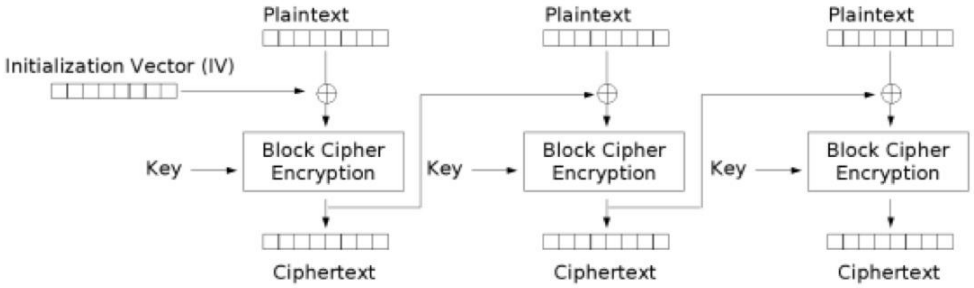


Electronic Codebook (ECB) mode encryption

- 평문을 고정된 크기의 블록으로 나눠주고 각각 독립적으로 암호화를 진행한다.
- 같은 키를 사용하여 동일한 평문 블록은 동일한 암호문 블록으로 변환된다.

장점	단점
● 병렬처리가 가능하다 ● 단일 블록의 손상이 다른 블록에 영향을 주지 않는다.	■ 블록 간의 상간관계를 고려하지 않아서 특정 유형의 공격에 취약할 수 있다.

5.2.2 CBC



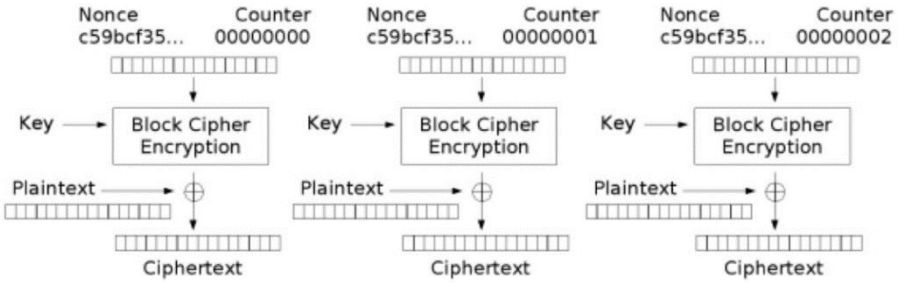
Cipher Block Chaining (CBC) mode encryption

- 이전 블록의 암호화 결과를 현재 블록의 암호화에 서로 연결하여 사용한다.
- 이전 암호문 블록을 XOR연산을 하고 이 결과를 암호화한다.
- 초기화 벡터가 사용된다.

장점	단점
● 같은 평문 블록이라도 매번 다	■ 오류 전파가 발생한다.

른 암호문 블록을 생성한다. ● 암호문 패턴을 분석하기 어렵게 만든다.	■ 병렬처리가 어렵다.

5.2.3 CTR

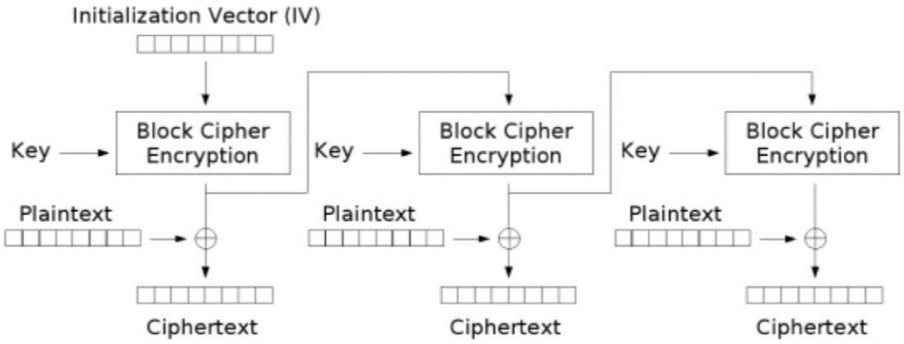


Counter (CTR) mode encryption

- 카운터라는 연속적으로 증가하는 값과 암호화 키를 사용하여 암호화 스트림을 생성한다.
- 평문 블록이 직접 암호화되지 않고 대신 카운터 값이 암호화되어 암호화 스트림을 형성한다.

장점	단점
● 높은 처리 속도 ● 높은 병렬 처리 능력 ● 오류 전파가 없다	● 동일한 카운터 값과 키 조합을 재사용하면 보안상의 위험을 초래할 수 있다. ● 데이터의 무결성을 보장하지 않는다.

5.2.4 OFB

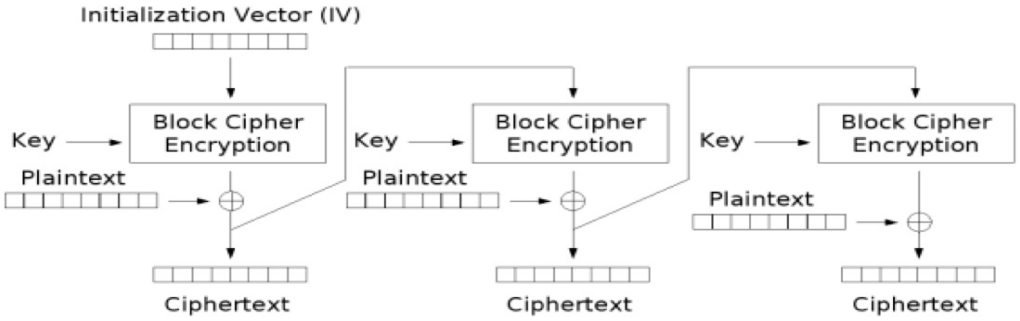


Output Feedback (OFB) mode encryption

- 블록 암호화의 운용 모드 중 하나로 스트림 암호의 특성을 가지고 있다.
- 암호화된 블록을 사용하여 다음 블록의 키 스트림을 생성한다.

장점	단점
● 오류 전파가 없다 ● 평문 크기에 상관없이 키 스트림을 생성할 수 있다.	■ 키 스트림의 재사용성은 보안상의 위험을 초래할 수 있다.

5.2.5 CFB

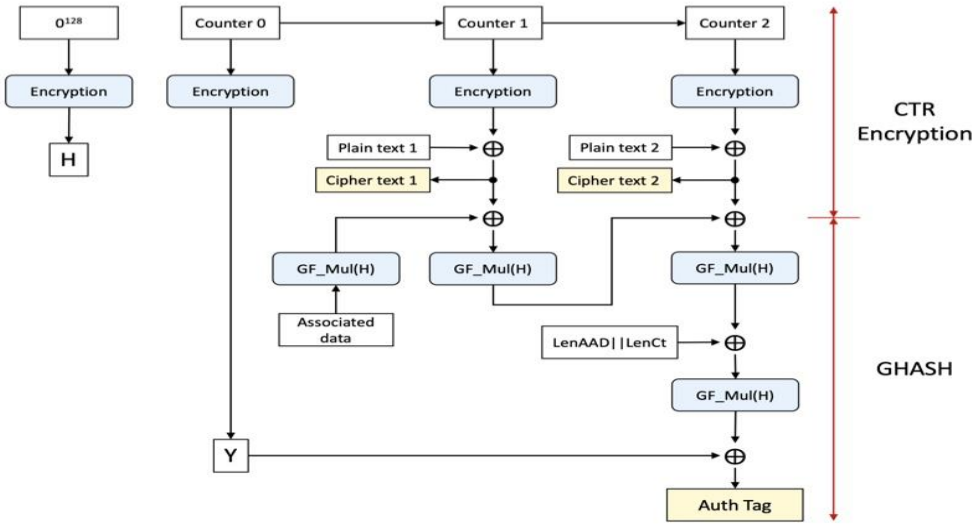


Cipher Feedback (CFB) mode encryption

- 스트림 암호와 유사한 방식으로 작동한다.
- 암호화 블록 체인의 일부분을 평문에 XOR연산하여 암호화를 수행한다.

장점	단점
● 암호화 복호화 과정이 스트림 암호와 유사하다. ● 블록 크기에 제한되지 않는 데이터 스트림 적용이 가능하다.	● 병렬처리가 어렵다. ● 데이터의 무결성을 보장하지 않는다.

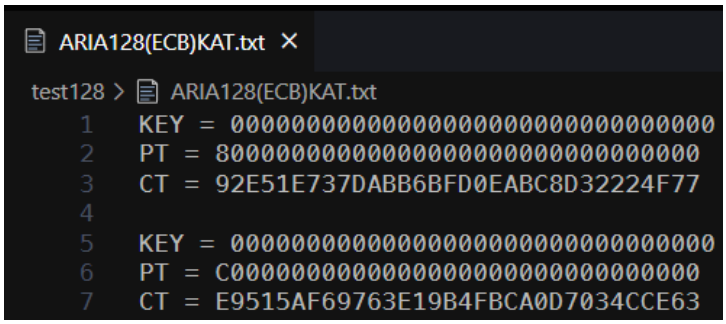
5.2.6 GCM



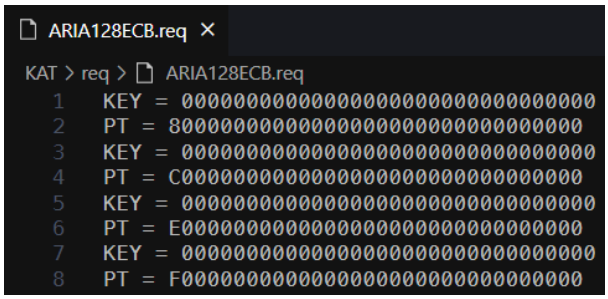
- CTR운용 모드를 이용해서 기밀성을 제공한다
- 암호화된 데이터와 AAD 데이터를 활용하여 GHASH함수 연산을 통해 무결성을 제공한다

장점	단점
● 병렬처리가 가능하다 ● 무결성을 제공한다.	● 키를 관리하는게 힘들다.

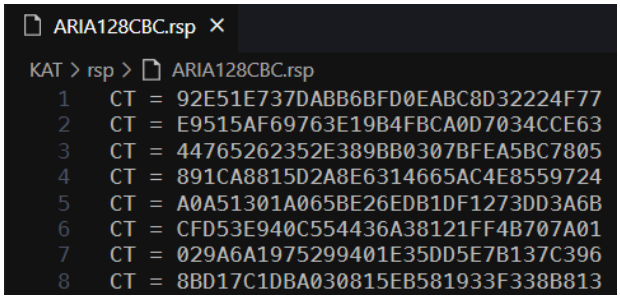
5.3. 작성한 보안알고리즘 TestVector로 검증.



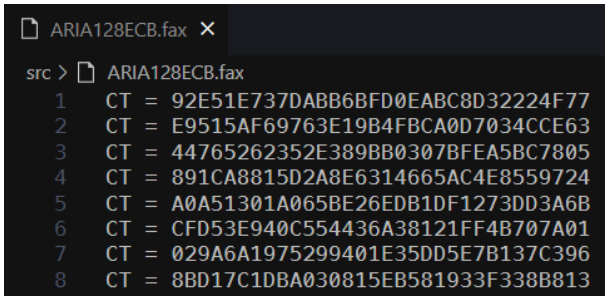
파싱 전 .txt 파일



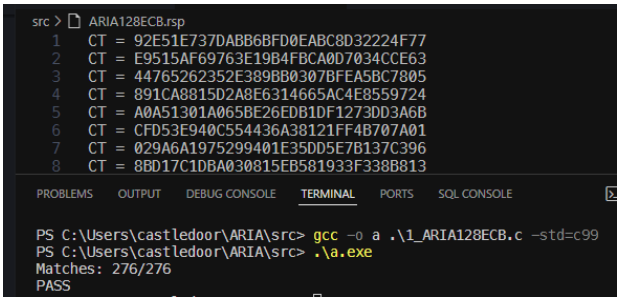
파싱 후 키와 평문 .req 파일



파싱 후 암호문 .rsp 파일



생성된 암호문 .fax 파일



테스트 결과값

위 그림과 같이 KEY, PlainText, CipherText로 구성된 기존의 .txt 파일을 파싱해서 키와 평문은 .req 파일에, 암호화를 정상적으로 수행했을 때의 CipherText 값은 .rsp 파일에 저장했습니다. 이후 .req 파일에 들어있는 KEY, PlainText값을 가지고 작성한 암호 알고리즘을 수행하여 만들어지는 CipherText들을 .fax 파일에 저장한 다음, 이 .fax 파일의 CipherText 값들을 .rsp 파일의 값과 비교하여 현재 작성한 알고리즘이 얼마나 정확한지를 판단하였습니다. CBC, CFB, OFB, CTR 운영 모드에서는 초기화 벡터 또는 카운터값이 필요하기 때문에 해당 값을 추가해서 진행하였습니다. **ARIA 알고리즘의 Test Vector는 KISA에 있는 것을 사용하였고, AES 및 ECC 알고리즘의 Test Vector는 NIST에 있는 것을 사용하였습니다.**

5.4. 알고리즘의 성능 측정

- **AES알고리즘에 대한 성능 측정**

키: 1111111111111111

평문: 0000000000000000

초기화 벡터: 난수로 생성

각 운영모드에 대한 CYCLECOUNTER

운영모드	ECB	CBC	CTR	OFB	CFB
CYCLECOUNTER	79031	79189	79281	79540	79526

전체적으로 비슷한 CYCLE수를 가지고 있는걸 볼 수 있었습니다.

- **ARIA알고리즘에 대한 성능 측정**

키: 000102030405060708090A0B0C0D0

평문: 303132333435363738393A3B3C3D3E3F

초기화 벡터: A0A1A2A3A4A5A6A7A8A9AAABACADAEAF

각 운영모드에 대한 CYCLECOUNTER

운영모드	ECB	CBC	CTR	OFB	CFB
CYCLECOUNTER	27598	28021	28024	28023	28152

- **ECC알고리즘에 대한 성능 측정**

키: 1234567890123456

평문: Hello, World!!!

초기화 벡터: A0A1A2A3A4A5A6A7A8A9AAABACADAEAF

초기화 벡터: 난수로 생성

운영모드	ECB	CBC	CTR	OFB	CFB
CYCLECOUNTER	17162	17093	17027	17073	18385

5.5. IAR툴을 이용한 최적화 수행

5.5.1. 복호화 때 사용할 구문을 암호화를 수행할 때 미리 저장해서 최적화

```
case CTR:
    for(int i = 0;i<block_size;i++)
    {
        for (int k = 0; k < 4; k++) {
            for (int j = 0; j < 4; j++) {
                save_reset[(k + (j * 4))] = reset_V[(k * 4) + j];
            }
        }

        aes_main(save_reset, expandedKey,nbrRounds);
        XORBlock(block[i], save_reset,BLOCKSIZE);
        increment_counter(reset_V,BLOCKSIZE);
    }

    break;
```

CTR운영모드 암호화

암호화 방식.

초기화 벡터 → 암호화 → 암호화된 초기화벡터 → 평문블록과 XOR 연산 → 암호화된 블록

복호화 방식

초기화 벡터 → 암호화 → 암호화된 초기화벡터 → 평문블록과 XOR 연산 → 복호화된 블록

이후 초기화벡터의 Counter부분에 값을 증가시키고 다시 반복해주면 된다.

여기서 암호화를 하는 방식과 복호화를 하는 방식을 비교했을 때 많이 중복되는 부분이 보일 것이다. 이런 중복되는 부분을 줄여서 CTR 운영 모드를 최적화를 해보고자 하였다.

```
case CTR:
    for(int i = 0;i<block_size;i++)
    {
        for (int k = 0; k < 4; k++) {
            for (int j = 0; j < 4; j++) {
                save_reset[(k + (j * 4))] = reset_V[(k * 4) + j];
            }
        }

        aes_main(save_reset, expandedKey,nbrRounds);
        for(int a = 0;a<16;a++)
        {
          operation_s[i][a] = save_reset[a];
        }
        XORBlock(block[i], save_reset,BLOCKSIZE);
        increment_counter(reset_V,BLOCKSIZE);
    }

    break;
```

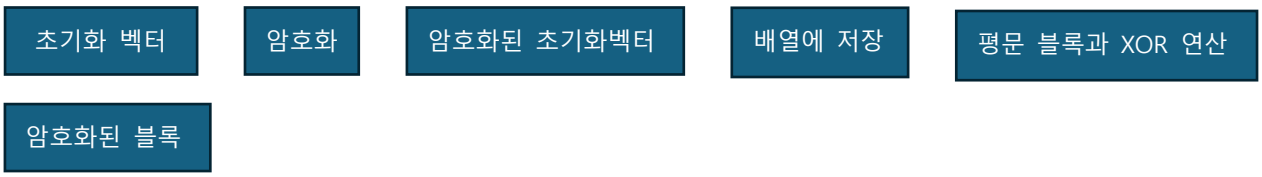
CTR운영모드 암호화 최적화

```
case CTR:
    for(int i = 0;i<block_size;i++)
    {

        XORBlock(block[i], operation_s[i],BLOCKSIZE);
        increment_counter(reset_V,BLOCKSIZE);
    }
    break;
```

CTR운영모드 복호화 최적화

암호화 방식.



복호화 방식

가장 처음 암호화를 진행할 때 평문에 XOR연산을 적용하기전에 나오는 초기화벡터에 대한 암호화값을 다른 배열에 저장을 해놓고 복호화를 진행할 때 이미 저장된 값을 가져와서 복호화를 진행하는 방식으로 최적화를 진행하였다. 암호화 과정을 복호화에서는 수행하지 않기 때문에 성능이 향상된다.

성능 측정

운영모드 = CTR, **알고리즘** = AES

Test Case.1

Key = 1111111111111111

Plaintext(평문) = 0000000000000000

Initialization Vector(초기화벡터) : 난수로 생성

	암호화	최적화 암호화	복호화	최적화 복호화
CYCLECOUNTER	79285	79613	79285	445

암호화를 진행하는 과정에서는 좀 더 많은 CYCLE수가 나오는 걸 확인할 수 있다. 하지만 복호화를 진행하는 과정에서는 굉장히 높은 성능을 가지는 걸 볼 수 있었다. **Plaintext의 양이 더 늘어나게 된다면 높은 성능을 발휘하게 된다.**

Test Case.2

Key = 1111111111111111

Plaintext(평문) = 00000000000000000000000000000000

Initialization Vector(초기화벡터) : 난수로 생성

	암호화	최적화 암호화	복호화	최적화 복호화
CYCLECOUNTER	158901	159089	158901	875

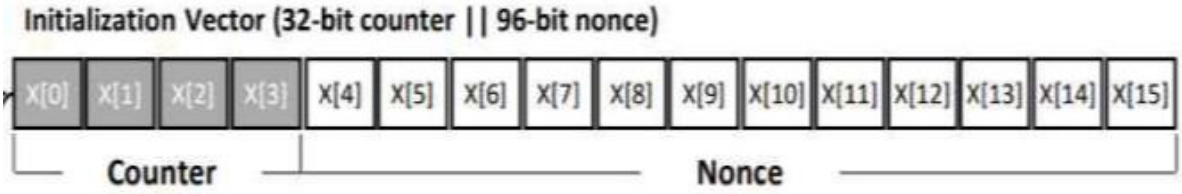
AES알고리즘은 평문을 128비트의 블록으로 나눠서 연산을 진행한다. 이때 부족하면 Padding을 이용해서 블록의 크기를 맞춰준다. 복호화를 하는 과정과 암호화를 하는 과정이 동일하기 때문에 TESTCASE에서는 암호화 복호화를 기준으로 CYCLECOUNTER의 개선을 보여줬습니다. 또한 연속적으로 동일한 초기화벡터를 사용해서 암호화를 진행한다면 처음 암호화할 때 오래 걸리고 이후의

암호화 과정에서는 저장한 값을 이용해서 빠른 암호화가 가능해집니다. **하지만 값들을 저장하는 공간을 따로 배정해야 하기 때문에 리소스를 사용하는 양이 늘어난다는 단점이 있다.**

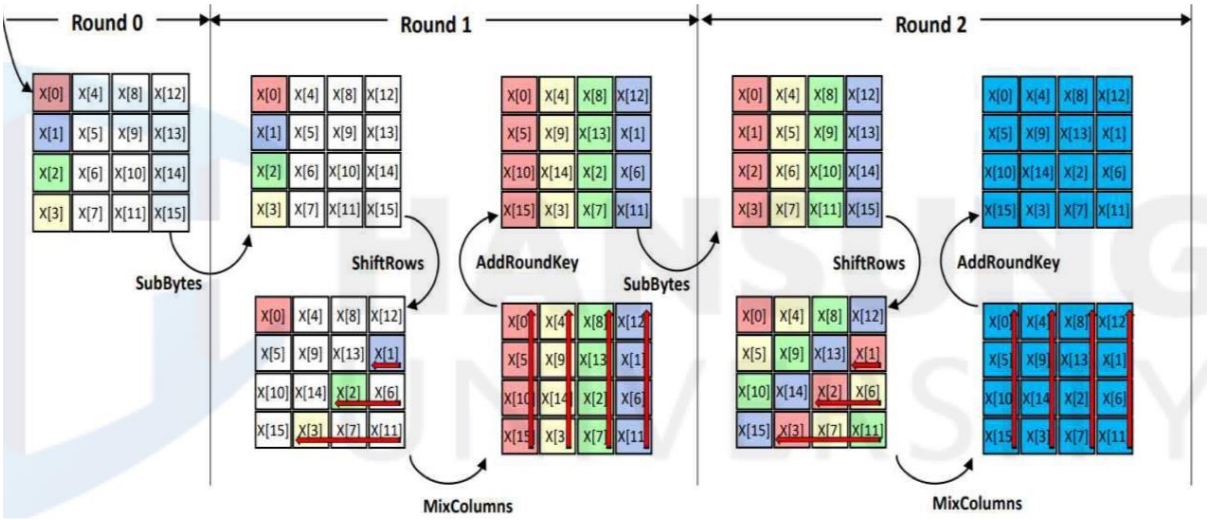
CTR 모드뿐만 아니라 다른 운영 모드에도 암호화할 때 사용했던 값을 미리 저장해놓으면 복호화를 진행할 때 더 좋은 성능으로 사용할 수 있을것으로 예상된다. **하지만 저장해놓은 값들은 암호화를 할 때마다 변하겠지만 암호문에 대한 키값을 저장하고 있기 때문에 보안상 문제가 발생할 수 있을 것으로 생각된다.** 보안성을 좀 더 높여줄 수 있는 방안을 생각해볼 필요가 있을 거 같다.

5.5.2. initialization vector를 이용한 사전연산기법으로 최적화 (CTR모드)

4.5.1에서 적용시켰던 최적화 방법은 복호화를 수행할 때 암호화를 수행하지 않고 복호화를 가능하게 하는 것으로 알고리즘의 수행시간을 줄이는 방법을 시도하였다. 이번에는 AES알고리즘을 수행하는 과정에서 최적화를 시도해보고자 한다.



그림에서 볼 수 있듯이 Count부분은 블록이 바뀔 때마다 x[0]부터 1씩 증가하는 부분이다. 그리고 증가한 초기화벡터를 암호화하고 다음 블록과 XOR연산을 해서 평문에 대한 암호문을 만들어준다.



그림을 보면 x[0], x[1], x[2], x[3]이 색칠된 부분이 각 열과 행에 영향을 주는 부분이다. 첫번째 라운드를 수행하는 과정을 보면 MixColumns을 수행하여도 통일된 색으로 열이 칠해지는 걸 볼 수 있다. 그리고 두번째 라운드를 시작하면 ShiftRows과정에서 기존의 Count를 저장했던 부분(x[0], x[1], x[2], x[3])이 MixColumns를 수행할 때 서로에게 영향을 주게 된다. 따라서 이번의 최적화 기법에서는 두번째 라운드에서 SubBytes 연산을 수행하고 난 이후의 각각의 열의 값을 배열에 저장해놓는 방식으로 최적화를 시도했다. 미리 저장하는 배열을 CTR_table라고 정의하겠다.

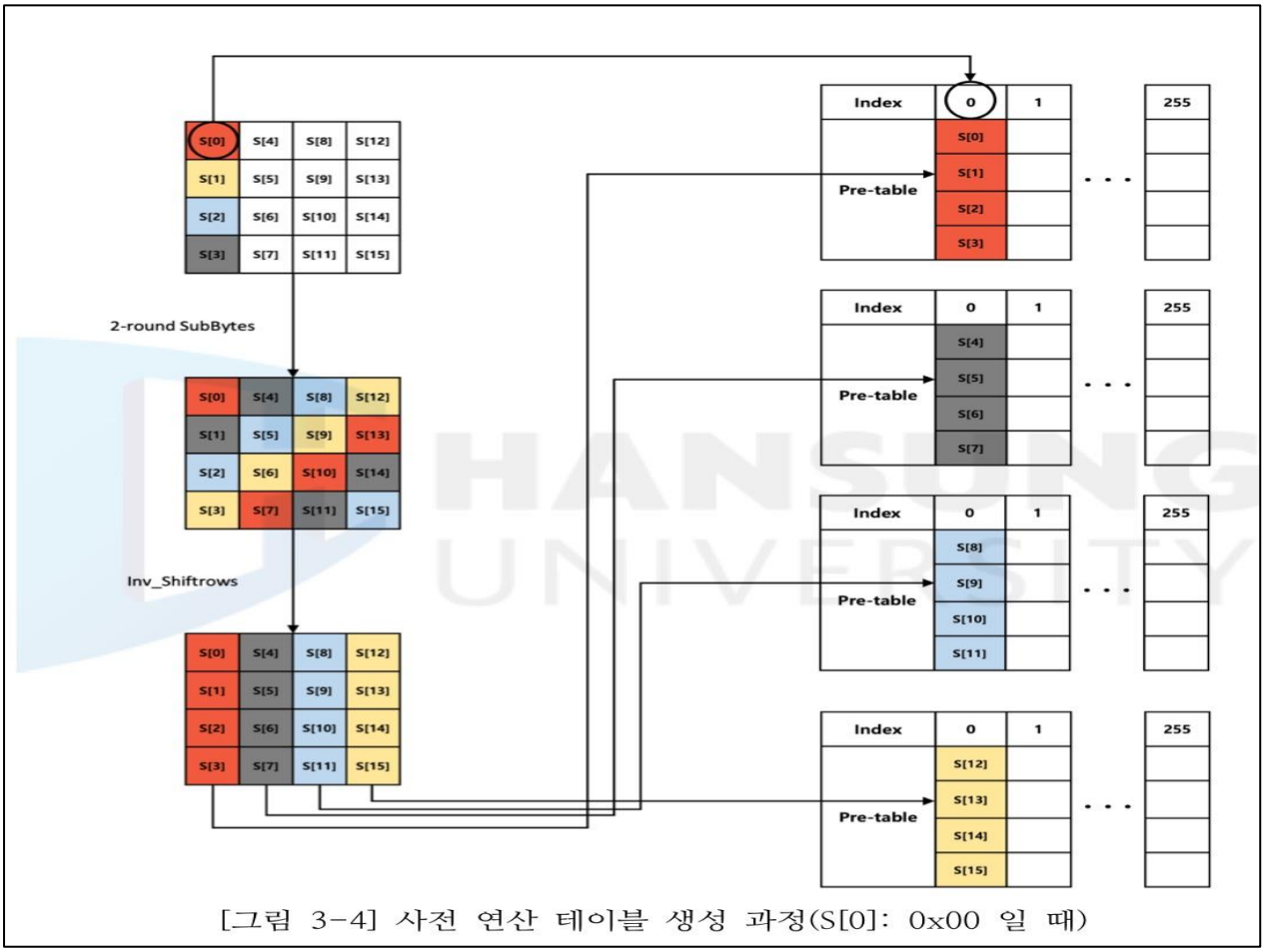
```
unsigned char roundKey[16];
unsigned char save_reset2[BLOCKSIZE];
for(int i = 0;i<blocknumber;i++)
{
    memcpy(save_reset2,state,16);
    createRoundKey(expandedKey , roundKey);
    addRoundKey(save_reset2, roundKey);
    createRoundKey(expandedKey , roundKey);
    aes_round(save_reset2, roundKey);
    subBytes(save_reset2);
        for(int k = 0;k<4;k++)
        {
          if(blocknumber == 1)
          {
            Pre_table1[k][state[0]] = state[(k*4)];
            Pre_table2[k][state[1]] = state[1+(k*4)];
            Pre_table3[k][state[2]] = state[2+(k*4)];
            Pre_table4[k][state[3]] = state[3+(k*4)];
          }
          else
          {
            if(blocknumber < 256)
                Pre_table1[k][state[0]] = state[(k*4)];
            else if(blocknumber < 512)
              Pre_table2[k][state[1]] = state[1+(k*4)];
            else if(blocknumber < 768)
              Pre_table3[k][state[2]] = state[2+(k*4)];
            else
              Pre_table4[k][state[3]] = state[3+(k*4)];
          }


        }
    increment_counter(state,BLOCKSIZE);
```

CTR_table을 미리 생성

```
unsigned char roundKey[16];
unsigned char state_save[16];
memcpy(state_save,state,BLOCKSIZE);
createRoundKey(expandedKey + 16 * 2, roundKey);
for(int i = 0;i<4;i++)
{
    state_save[(i*4)] = Pre_table1[i][state[0]];
    state_save[1+(i*4)] = Pre_table2[i][state[1]];
    state_save[2+(i*4)] = Pre_table3[i][state[2]];
    state_save[3+(i*4)] = Pre_table4[i][state[3]];
}
shiftRows(state);
mixColumns(state);
addRoundKey(state, roundKey);
for(int i = 3;i<nbrRounds;i++)
{
    createRoundKey(expandedKey + 16 * i, roundKey);
    aes_round(state, roundKey);
}
createRoundKey(expandedKey + 16 * nbrRounds, roundKey);
subBytes(state);
shiftRows(state);
addRoundKey(state, roundKey);
}
```

AES알고리즘 수행 변경



[그림 3-4] 사전 연산 테이블 생성 과정(S[0]: 0x00 일 때)

CTR_table을 저장할때는 바이트에 저장된 카운터값을 Index로 이용해서 열의 값을 저장해준다

성능 측정

운영모드 = CTR, **알고리즘** = AES

Test Case.1

Key = 1111111111111111

Plaintext(평문) = 0000000000000000

Initialization Vector(초기화벡터) : 난수로 생성

	기존 알고리즘	4.5.1 적용 알고리즘	4.5.2 적용 알고리즘
암호화	79285	79613	85052
복호화	79285	445	70300

Test Case.2

Key = 1111111111111111

Plaintext(평문) = 00000000000000000000000000000000

Initialization Vector(초기화벡터) : 난수로 생성

	기존 알고리즘	4.5.1 적용 알고리즘	4.5.2 적용 알고리즘
암호화	158901	159089	164565
복호화	158901	875	140622

Test Case.3

Key = 1111111111111111

Plaintext(평문) = 0000000000000000000000000000000000000000000000000000000000000000

Initialization Vector(초기화벡터) : 난수로 생성

	기존 알고리즘	4.5.1 적용 알고리즘	4.5.2 적용 알고리즘
암호화	317378	317814	323815
복호화	317377	1738	281190

4.5.1을 적용한 알고리즘에 비해 복호화를 하는 과정에서 굉장히 느린 걸 볼 수 있었다. 하지만 기존의 알고리즘 보다는 복호화를 하는 과정이 빠른걸 알 수가 있었고, **평문의 크기가 커질수록 암호화를 할때와 복호화를 할때의 연산양이 줄어드는 걸 확인할 수 있었다.** 또한, 4.5.1에서 우려했었던 보안상의 문제를 4.5.2에서는 복호화 키값을 저장하고 있는 것이 아니라 저장한 값으로 암호화를 수행하고 복호화를 진행하기 때문에 보안성은 더 높은 알고리즘이라고 생각할 수 있다.

5.6. QNX취약점 분석

0. 용어 설명

CVE(공통 취약점 및 노출, Common Vulnerabilities and Exposures)

CWE(공통 약점 열거, Common Weakness Enumeration)

1. CVE-2021-22156

설명: 영향을 받는 BlackBerry® QNX 소프트웨어 개발 플랫폼(SDP) 버전 6.5.0SP1 및 이전 버전, QNX OS for Medical 1.1 및 이전 버전, QNX OS for Safety 1.0.1 및 이전 버전의 C 런타임 라이브러리의 calloc() 함수에 정수 오버플로 취약점이 있으며, 이를 통해 공격자가 잠재적으로 서비스 거부 공격을 수행하거나 임의 코드를 실행할 수 있다.

사례: 위의 취약점으로 인해 악의적인 공격자가 자동차, 의료 및 산업 장비를 포함한 다양한 제품을 불구로 만들고 제어할 수 있는 것으로 나타났다.

블랙베리의 QNX 기술은 전 세계적으로 항공 우주, 방위, 자동차, 의료, 로봇 공학 등 광범위한 산업 분야에서 195만 대 이상의 차량과 임베디드 시스템에 사용되고 있다. 원격 공격자는 취약점 CVE-2021-22156을 악용하여 서비스 거부 상태를 유발하거나 영향을 받는 장치에서 임의의 코드를 실행할 수 있다.

2. CWE-190: 정수 오버플로우 또는 랩어라운드

설명: 이 제품은 계산을 수행하는데, 이 과정에서 결과 값이 항상 원래 값보다 클 것이라고 가정할 때 정수 오버플로우나 랩어라운드가 발생할 수 있다. 이는 정수 값이 해당 표현 방식에 저장할 수 있는 최대치를 초과하여 증가할 때 발생한다. 이 경우, 결과 값은 매우 작은 수나 음수로 변할 수 있다.

약점 예시

다음 이미지 처리 코드는 이미지에 대한 테이블을 할당한다.

예시 언어: C (잘못된 코드)

```
img_t table_ptr; /*각각 10kB 크기의 img 데이터를 담은 구조체*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
...
```

이 코드는 num_imgs 크기의 테이블을 할당하려고 하지만 num_imgs가 커짐에 따라 목록 크기를 결정하는 계산이 결국 오버플로된다(CWE-190). 그러면 대신 매우 작은 목록이 할당된다. 후속 코드가 목록이 num_imgs 길이인 것처럼 작동하면 많은 유형의 범위를 벗어난 문제가 발생할 수 있다(CWE-119).

단계별 약점 완화책

요구 사항: 모든 프로토콜이 엄격하게 정의되어 모든 범위를 벗어난 동작을 쉽게 식별할 수 있게 프로토콜을 엄격히 준수하도록 요구하고 이러한 약점이 발생하지 않도록 하는 언어를 사용하거나 이러한 약점을 피하기 쉽게 만드는 구성을 제공한다. 가능하다면 자동 범위 검사를 수행하는 언어나 컴파일러를 선택한다.

아키텍쳐 및 디자인: 예상치 못한 결과 없이 숫자를 쉽게 처리할 수 있는 라이브러리나 프레임워크를 사용한다. 예로는 SafeInt(C++) 또는 IntegerLib(C 또는 C++)와 같은 안전한 정수 처리 패키지가 있다. [REF-106]

구현:예상 범위 내에 있는지 확인하여 모든 숫자 입력에 대한 입력 검증을 수행한다. 입력이 예상 범위에 대한 최소 및 최대 요구 사항을 모두 충족하도록 적용한다.

가능하면 부호 없는 정수를 사용한다. 그러면 정수 오버플로에 대한 검증을 수행하기가 더 쉬워진다. 부호 있는 정수가 필요한 경우 범위 검사에 최소값과 최대값이 모두 포함되어 있는지 확인한다.

프로그래밍 언어의 기본 표현과 숫자 계산과 상호 작용하는 방식을 이해한다.(CWE-681). 바이트 크기 불일치, 정밀도, 부호/무부호 구분, 잘림, 유형 간 변환 및 캐스팅, "숫자가 아닌" 계산, 언어가 기본 표현에 비해 너무 크거나 작은 숫자를 처리하는 방식에 주의한다. [REF-7]

또한 숫자 표현에 영향을 줄 수 있는 32비트, 64비트 및 기타 잠재적인 차이점도 신중하게 고려한다.

3. CVE-2019-8998

설명: BlackBerry QNX 소프트웨어 개발 플랫폼 버전 6.5.0 SP1 및 이전 버전의 procfs 서비스(/proc 파일 시스템)에서 권한의 로컬 에스컬레이션으로 이어질 수 있는 정보 공개 취약점을 통해 공격자는 선택한 프로세스 주소 공간에 대한 무단 액세스를 얻을 수 있다.

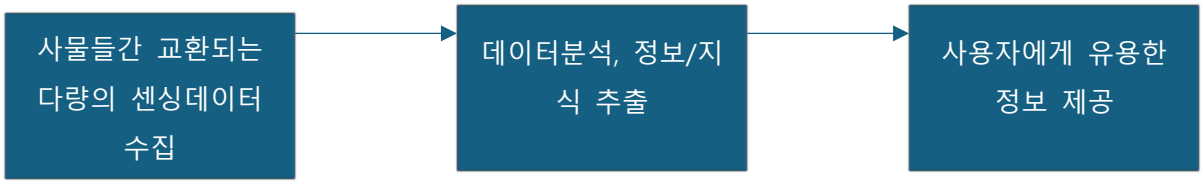
사례: 미국 국토안보부 산하 사이버보안 및 인프라 보안국(CISA)은 지난주 기관들에 Tridium의 Niagara 제품이 임베디드 장치용 BlackBerry QNX 운영 체제의 두 가지 취약점에 영향을 받는다고 알렸다. Honeywell의 자회사인 Tridium에서 배포한 QNX 운영 체제 이미지는 최근 공개된 몇 가지 취약점의 영향을 받는다. 보안 홀은 Niagara AX 3.8u4, Niagara 4.4u3 및 Niagara 4.7u1에 영향을 미친다. 두 가지 취약점 중 더 심각한 취약점은 CVE-2019-8998로 추적되었으며 CVSS 점수는 7.8이었다. BlackBerry가 7월에 발표한 권고안 에 따르면, CVE-2019-8998은 procfs 서비스와 관련

된 정보 공개 문제이며 로컬 권한 상승에 악용될 수 있다.
공격자가 성공하기 위한 요구사항: 이러한 취약점을 악용하려면 공격자가 사용자 권한을 사용해 운영 체제에 악성 애플리케이션을 만들어 배포해야 한다. 성공적인 공격자는 선택한 프로세스 주소 공간에 대한 무단 액세스를 얻을 가능성이 있다.
해결 방법: 프로덕션 운영 체제의 개발자와 관리자는 procnto -u umask 옵션을 설정하여 이 공격을 제한할 수 있다

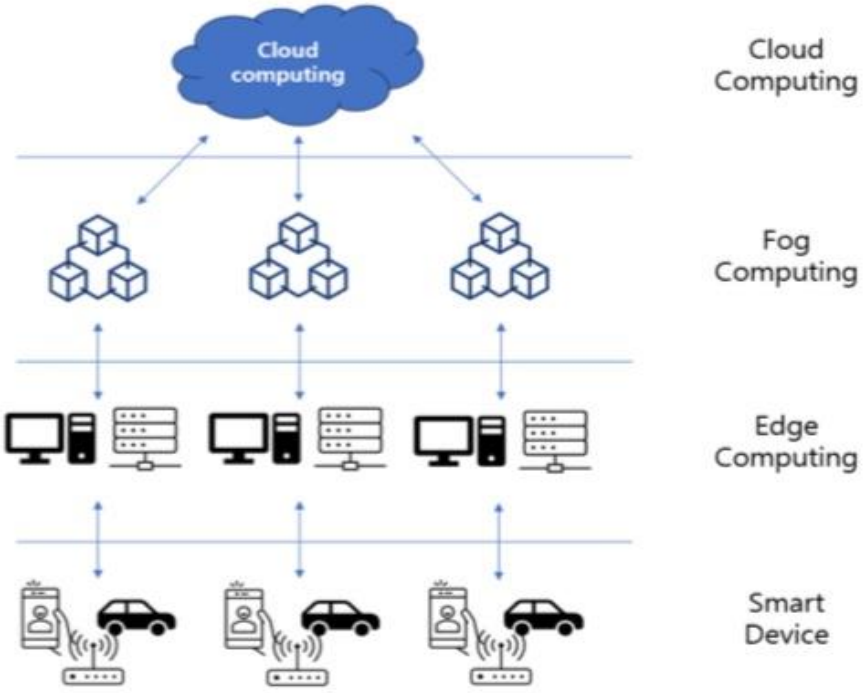
6. 자문의견서에 대한 보완

- **RTOS내의 코드 최적화 수행이 임베디드 보안에 미치는 영향**

사물인터넷 기술은 사용자에게 개인 맞춤형 서비스를 제공하면서 삶의 질을 향상시키고 있다. 심지어 요즘에 출시되는 스마트폰은 내장형AI를 가지고 있으며 오프라인 상황에도 전화 통화를 할 때 실시간 번역을 해주는 시스템이 내장되어 있는 스마트폰도 출시되고 있다. 이외에도 임베디드, IOT기술은 우리의 일상속에 함께 하고 있다. 스마트홈, 농업분야, 의료분야 등등 다양한 곳에서 기술이 사용되고 있다. IOT기술의 방식을 간략하게 설명하면



이런 과정을 거쳐서 사용자에게 유용한 정보를 제공하게 된다. 하지만 다량의 센싱 데이터는 개인의 민감한 정보를 가지고 있다. 그래서 누군가가 정보를 가로채게 된다면 사용자에게 큰 피해를 줄수도 있다. 그래서 보안을 강화하기 위해서 평문으로 보내는 게 아닌 암호알고리즘을 거쳐서 각 단계를 진행해야 한다. 하지만 고성능 컴퓨터에서 사용하는 알고리즘을 그대로 임베디드 시스템 상에 올리기에는 무리가 존재한다. 왜냐하면 고성능 시스템에 비해서 제한



된 리소스에서 암호화를 수행해야 하기 때문이다. IOT아키텍처를 참고하면
가장 하단에 존재하는 Smart Device Layer의 경우 낮은 용량의 장치로 분류가 되는데 스마트 센서부터 Arduino에 이르기까지 다양하며

플래시 메모리: 8KB ~ 8MB

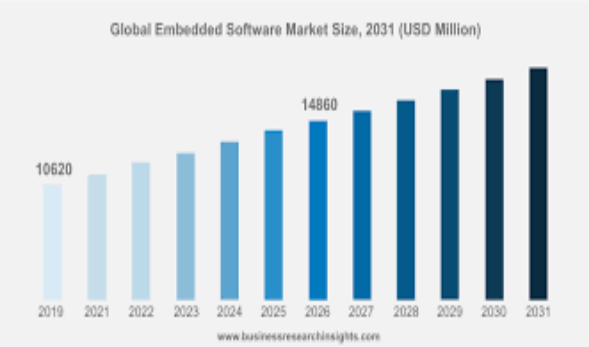
RAM: 512B ~ 256MB(라즈베리 파이 아두이노 새로운 버전 제외)

CPU: 0MHz ~ 400MHz

정도의 성능을 보여준다. 따라서 보안 매커니즘은 가벼우면서 효율적으로 만들어져야 한다. 물론 고성능 컴퓨터에 올라가는 보안 알고리즘의 경우 높은 보안성을 제공하고 있겠지만 리소스가 제한된 장치에서는 실행하기가 어렵거나 실시간 시스템에서는 불가능 할 수 있다. 따라서 보안 알고리즘의 경량화는 필수적으로 보인다.

경량화를 얻을 수 있는 이점

1. 기존의 알고리즘의 메모리 사용량을 줄여서 더 작은 시스템에 고성능의 암호 알고리즘을 넣는 것이 가능해진다.
2. 경량화를 통한 연산의 속도를 감소시켜 RTOS시스템의 응답속도를 개선이 가능하다 응답속도를 개선해서 보안 위협에 대응하는 시간을 단축할 수 있다.
3. 경량화를 통해서 암호 알고리즘을 수행하는 동안의 사용되는 전력의 양을 줄일 수 있다.



출처 : Business Research Insight

위의 자료에서 알 수 있듯이 세계 임베디드 시장이 커지는 것에 맞춰서 임베디드 보안 상품에 대해서도 증가할 것이라는 전망이 있다. 또한 NIST에서도 경량암호 공모사업을 꾸준히 함으로써 다양한 경량 암호 알고리즘을 개발하는데 힘을 쓰고 있다. 이처럼 암호 알고리즘의 경량화는 중요하다는 걸 알 수 있다.

7. 참고문헌

참고논문

- RISC-V상에서의 Fixslicing AES CTR 운용 모드 최적화 구현 – 엄시우
  A. initialization vector를 이용한 사전연산기법으로 최적화 (CTR모드)이미지 출처
- 저전력 프로세서 상에서의 AES-GCM 최적화 구현 – 김경호
  A. initialization vector를 이용한 사전연산기법으로 최적화 (CTR모드)이미지 출처
- 저전력 IOT 장치를 위한 경량 블록 암호화 알고리즘 운용 방법에 대한 연구 – 홍재완
  A. 운영모드 이미지 출처
- 32-BIT RISC-V상에서의 LEA 경량 블록 암호 GCM 운용 모드 구현 – 엄시우, 김현준, 심민주, 송경주, 서화정