최종 보고서

Airplane / 300km

1. 개발진 소개


- 이름 : 위승주, 이재석
- 학년 : 3학년
- 소속 : 암호 최적화 및 응용 연구실
- 특이사항 : 복학하고 돌아온 귀요미들
- 현재 하고 있는 일 : GPU연구

2. 개발 목표와 전략

-개발목표

범용적 환경의 동적 연산 라이브러리와 정적 연산을 통한 암호 라이브러리(ECDH)

-역할 분배 및 추진 전략

이재석	• 동적 라이브러리 코드 만들기 • 정적 라이브러리 코드를 파이썬을 활용한 검증툴로 검증하기 • 코드 주석 달기 • PQClean의 AES-DRBG를 사용가능한 코드로 바꿔오기 • ECDH를 위한 정적 Big number 라이브러리 코드 만들기 • Static Big number Library를 활용한 ECADD 및 ECDBL구현 • CMake를 이용한 전체적 라이브러리 만들기
위승주	• 동적 라이브러리 코드 만들기 • 동적 라이브러리 코드를 파이썬을 활용한 검증툴로 검증하기 • Valgrind를 이용한 메모리 Leak 찾기 • ECDH를 위한 정적 Big number 라이브러리 코드 만들기 • Static Big number Library를 활용한 ECADD 및 ECDBL구현 • Static Big number Library를 활용한 ECADD 및 ECDBL구현

라이브러리 구현의 경우 대부분을 같이 하였습니다. 그 외에 필요한 랜덤값을 위한 것이나 메모리 누수 확인, make파일 만들기 등을 역할을 나누어 서로 열심히 하였습니다.

3. 개발 일지

주차	일정
1주차	• Orientation
2주차	• (학회 발표로 인한 수업 미참석)
3주차	• 인턴과 첫 만남 & 회의 • Basic operation 동적 구현(20%)

	• Cmake & 필요한 툴 구성
4주차	• Basic operation 동적 구현(30%) • 검증 툴 구현 및 적용 시작
5주차	• Basic operation 동적 구현(30%) • 동적 Basic operation 검증(30%) • 정적 Big number 구현(512bit) (20%)
6주차	• Basic operation 동적 구현(80%) • 동적 Basic operation 검증(80%) • 정적 Big number 구현(512bit) (50%) • AES-DRBG 코드 가져오기(사용가능하게) • AES-DRBG 활용한 Rand값 생성 함수 제작
7주차	• Basic operation 동적 구현 완료(99%) • Add, Sub 동적 구현(40%) • 정적 Basic operation 구현(90%) • 정적 Basic operation 검증(90%) • 정적 Add, Sub 구현(100%) • 정적 Add, Sub 검증(100%)
8주차	• 중간 보고서 제출 • 동적 Add, Sub 구현(100%) • 동적 Add, Sub 검증(100%) • 동적 school book mul 구현(90%) • 정적 school book mul 구현(100%) • 정적 school book mul 검증(100%)
9주차	• 동적 Basic operation Memory Leak검사 • 동적 Add, Sub Memory Leak검사 • 동적 school book mul 구현(100%) • 동적 school book mul 검증(100%) • 정적 karatsuba mul 구현(100%) • 정적 karatsuba mul 검증(100%)
10주차	• 정적 역원 알고리즘 구현 및 검증 • 동적 improved school book mul 구현(100%) • 동적 karatsuba mul 검증(100%)
11주차	• 정적 Jacobian좌표 상 구현 및 검증 • 동적 라이브러리 전체적 Memory Leak 검사 • 동적 나눗셈 알고리즘 구현 및 검증
12주차	• 동적 라이브러리 및 정적 라이브러리 코드 속도 비교 (곱셈: School Book vs Karatsuba, 역원 : FLT vs EEA) • 정적 Jacobian좌표 상 구현 및 검증
13주차	• ECADD, ECDBL 구현 및 검증 • 동적 Barret Reduction 구현 및 검증
14주차	• 동적 라이브러리 마무리 • ECDH 로직 정적 구현
15주차	• 마무리 및 프로젝트 발표 & (회식)

4. 개발 라이브러리 소개

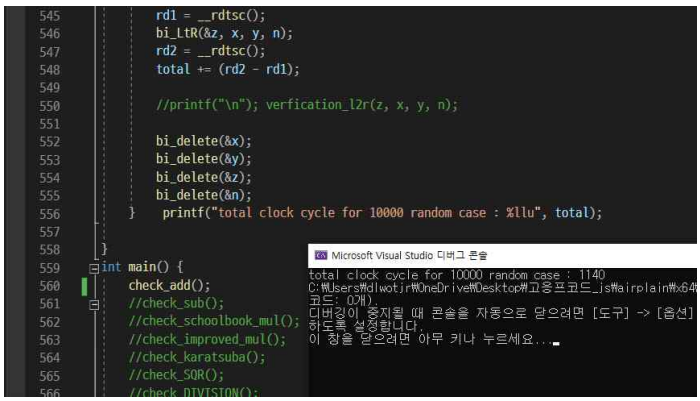
-동적 라이브러리의 경우 평가표에 보다 자세히 나와있습니다. 참고 부탁드립니다

-동적 빅넘버 라이브러리

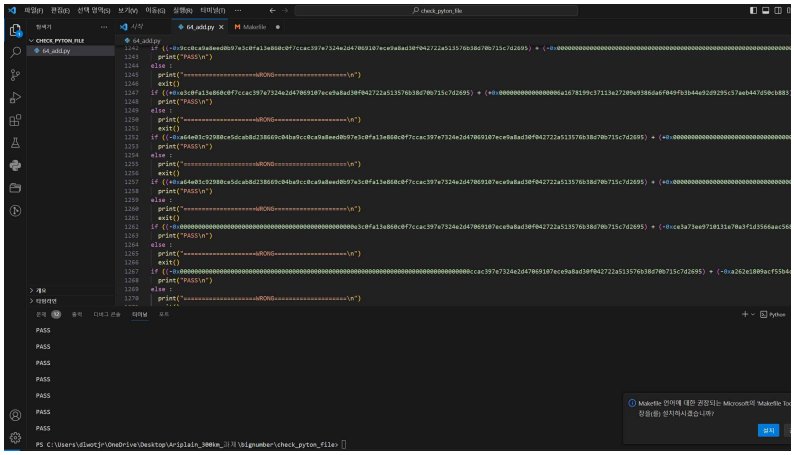
- ➢ 리눅스(gcc, g++) 및 윈도우(msvc) 사용 가능
- ➢ 구동 함수
  - ✓ ADD
  - ✓ SUB
  - ✓ MUL (school book, improved, Karatsuba)
  - ✓ SQR
  - ✓ DIV (binary)
  - ✓ REDC (barrett)
  - ✓ EXP (L2R)
- ➢ 검증기법 : Phyton으로 검증

-구동 시 결과값 및 성능측정/ 검증(예시 : add)

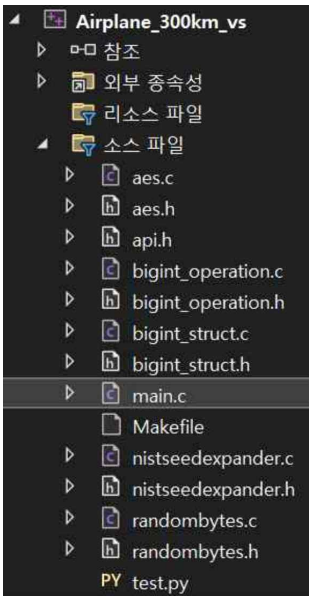
add함수에서의 결과값 화면



add함수에서의 성능 측정



add함수에서의 파이썬 검증



구동 방법 - <visual studio>

```
sj@ubuntu:~/Downloads/js_test$ make all
gcc     -g -I./include ./bigint_operation.c ./nistseedexpander.c ./aes.c ./bigint_struct.c ./randombytes.c main.c -o test
sj@ubuntu:~/Downloads/js_test$ ./test
```

구동 방법 - <리눅스 - g++, gcc>

-정적 ECDH 라이브러리

1. ECDH 수행방법 소개

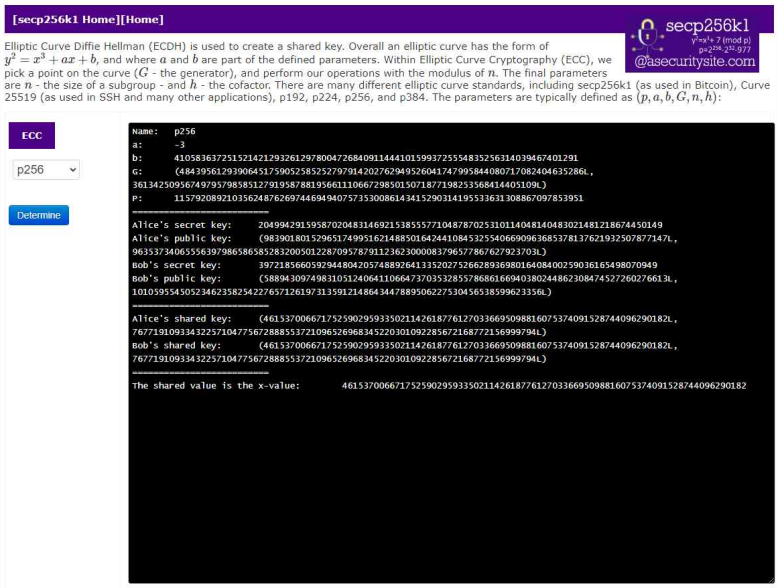
컴파일러 : g++

실행환경 : vscode 2022

2. 실행 결과물(비교 결과물)

```
PS C:\Users\coala_guest\Desktop\WI\Class\3-2\application\ECC> g++ .\ECC_two_struct.c .\bignum.c
PS C:\Users\coala_guest\Desktop\WI\Class\3-2\application\ECC> .\a.exe
[Alice's secret key]
2d5244073e432d4a3a2b196ec95eba75aa6f0425c7806f678e3055b9899adee5
[Alice's public key]
d986de55e7fdf4dacdad6f9ea8d4e11bb3b54ef4d4ff3054992cf9fa6f374f1b
d50647e6606a1c526503fe593c542336e6d6accdd5f5091711e203d6f05288f7

[Bob's secret key]
08c82d6ad6e55cb6ad7b8572c071486d455fbf1ea7b2dcf49ef9fafd5fbde7a5
[Bob's public key]
8234ff7fa54856a5d9a14e028efb694693bd487c48bf37a6df501b28ead2f785
df6dae4b90c29fa19f3ec62ae4fa3eb3ce999f7d467144f278d6c904b6ef8abc

[Alice's shared key]
660a119feaae90989aebbc228c4f27e3ea7e85cbed760a1433a8628ceb90d586
a9bb5c9a57d7e80b8218c60b0853fd26123b96fa96baac9be513ec2b265b4872

[Bob's shared key]
660a119feaae90989aebbc228c4f27e3ea7e85cbed760a1433a8628ceb90d586
a9bb5c9a57d7e80b8218c60b0853fd26123b96fa96baac9be513ec2b265b4872
PS C:\Users\coala_guest\Desktop\WI\Class\3-2\application\ECC>
```

실제 실행 결과



비교 매체 (https://asecuritysite.com/secp256k1/ecdh3)

해당 비교 메체는 ECDH의 결과값을 보여줍니다. 하지만 0x가 앞에 붙지 않은 모든 경우는 정수값으로 나와 있으며 해당 마지막 결과를 pyton을 통하여 hex변환시에 동일함을 볼 수 있습니다. 아쉽게도 키 생성은 하지 못하였으며 해당 비교매체의 alice의 비밀키와 공개키, bob의 비밀키와 공개키를 인자로 받아 jacobian좌표상의 Right to Left(ECC scala multiplication)을 통해 해당 값이 동일함을 볼 수 있었습니다. 해당 함수의 결과값으로 아핀 좌표계에서 (x,y)의 값이 나오며 각 x와 y중 x를 선택하여 shared value로 사용함을 볼 수 있습니다.

3. 내부 코드

사실 main을 보시면 한 것이 없어 보일수 있지만 사실 매우 많은 것이 들어있습니다.

정적인 상태의 덧셈, 뺄셈, 곱셈에서도 os와 ps 그리고 카라츄바 알고리즘이 있으며 제곱연산

도 구현되어 있고 역원계산으로 FLT(페르마 마지막 정리), EEA(확장 유클리드 알고리즘) 또한 Reduction방법으로는 ECC의 최적화 구현이 Fast reduction을 비롯해 몽고메리 리덕션을 구현하였습니다. 그리고 지수승에 해당하는 Left to Right, Right to Left의 함수가 구현되어 있습니다.

```
void add_int(const uint32_t a, const uint32_t b, uint32_t *result, int *carry);
void add_bigint(uint32_t a[SIZE], uint32_t b[SIZE], uint32_t result[SIZE]);
void add_bigint_16(uint32_t a[SIZE*2], uint32_t b[SIZE*2], uint32_t result[SIZE*2]);
void sub_int(const uint32_t a, const uint32_t b, uint32_t *result, int *borrow);
void sub_bigint(uint32_t a[SIZE], uint32_t b[SIZE], uint32_t result[SIZE]);
void mul_bigint_os(const uint32_t a[SIZE], const uint32_t b[SIZE], uint32_t result[SIZE*2]);
void mul_bigint_ps(const uint32_t a[SIZE], const uint32_t b[SIZE], uint32_t result[SIZE*2]);
void sqr_bigint(const uint32_t a[SIZE], uint32_t result[SIZE*2]);
void redn_bigint_fast(const uint32_t a[SIZE*2], uint32_t result[SIZE]);
int compare_bigint(const uint32_t a[SIZE], const uint32_t b[SIZE]);
void redn_bigint_mont(const uint32_t a[SIZE*2], uint32_t result[SIZE]);
void div_bigint_by_bitshift(const uint32_t a[SIZE], uint32_t b[SIZE], const uint32_t loop_cnt);
void mul_bigint_by_bitshift(const uint32_t a[SIZE], uint32_t b[SIZE], const uint32_t loop_cnt);
void inv_bigint_EEA(const uint32_t a[SIZE], uint32_t result[SIZE]);
void sqr_bigint_loop(const uint32_t a[SIZE], uint32_t result[SIZE], const uint32_t loop_cnt);
void inv_bigint_FLT(const uint32_t a[SIZE], uint32_t result[SIZE]);
void mul_and_redn_bigint(uint32_t a[SIZE], uint32_t b[SIZE], uint32_t result[SIZE]);
void sqr_and_redn_bigint(uint32_t a[SIZE], uint32_t result[SIZE]);
void mul_bigint_LtR(uint32_t a[SIZE], const uint32_t b[SIZE], uint32_t result[SIZE]);
void mul_bigint_os_4(const uint32_t a[SIZE/2], const uint32_t b[SIZE/2], uint32_t result[SIZE*2]);
void mul_bigint_os_5(const uint32_t a[SIZE/2+1], const uint32_t b[SIZE/2+1], uint32_t result[SIZE*2]);
void add_bigint_4(const uint32_t a[SIZE/2], const uint32_t b[SIZE/2], uint32_t result[SIZE/2+1]);
int compare_bigint_16(const uint32_t a[SIZE*2], const uint32_t b[SIZE*2]);
void sub_bigint_16(uint32_t a[SIZE*2], uint32_t b[SIZE*2], uint32_t result[SIZE*2]);
void mul_bigint_karatsuba(const uint32_t a[SIZE], const uint32_t b[SIZE], uint32_t result[SIZE*2]);
```

bignum.h파일 내부 코드

4. 성능 측정

-노트북으로 진행하여 느린 것을 볼 수 있지만 동적 라이브러리와의 비교를 위해서는 동일 환경에서 다시 측정하여 평가표에서 비교를 했습니다

성능측정 함수 : __rdtsc

테스트 벡터 : 20000개

연산	Clocks
ADD	202868721
SUB	206745034
MUL - OS	267455255
MUL - PS	258998490
SQR	207686776
REDN - Fast	327870068
REDN - Mont	606842313
REDN - Mont (Fast)	563012365
INV - EEA	9634145744
INV - FLT	10349497920

성능 측정표

4-1) 곱셈연산이 os와 ps의 연산속도는 유의미한 차이를 보이지 않았습니다

4-2) 곱셈연산과 제곱 연산 사이에는 유의미한 차이가 확인되었습니다.

4-3) ECC 최적화로 구현된 Reduction연산인 Fast의 경우 몽고메리와 몽고메리 Fast보다 유의미하게 빨랐습니다.

4-4) 역원계산에서 EEA와 FLT 사이에는 큰 차이가 없었습니다.

5. 라이브러리 활용 사례

-활용 사례 1

암호와 수학을 동시에 공부할 수 있는 라이브러리!

좋고 빠른 연산 라이브러리는 많습니다. 해당 라이브러리보다 빠르거나 더 많고 좋은 연산을 지원하기는 무리가 있다고 생각합니다. 하지만 저희의 동적 라이브러리는 누구보다 슈도코드를 따라가며 정직하게 짰고 정적 라이브러리는 더 다양한 함수와 실제 암호에서 쓰이는 함수들 그리고 정적 구현 방법을 볼 수 있습니다.

예를 들어 스쿨북 곱셈을 배운 암호학과 1학년 학생은 슈도 코드를 보고 조금 쉽다고 느낄 수 있지만 막상 코드를 짜려고 하면 어려움을 느낄 것입니다. 따라서 정직하게 따라간 코드를 보고 어떤 부분을 어떻게 따라가야 되는지를 배우고 동적 할당과 메모리 관리의 개념에 대해 배운 후 정적 라이브러리를 봄으로써 암호에 사용되는 스쿨북 곱셈만이 아닌 카라츄바, ps곱셈, 추후에는 톰쿡, NTT등을 공부하며 배울 수 있습니다.

-활용 사례 2

ECDH 정적 라이브러리를 통하여 TLS1.3등 프로토콜을 만들 수 있습니다.

6. 수익 모델

➢ 암호 및 코딩을 공부하는 학생에게 직접 쓴 수학에 관한 교제 및 코드까지!!

➢ 암호에 필요한 수학, 코딩(동적, 정적)이 다 들어가있다!

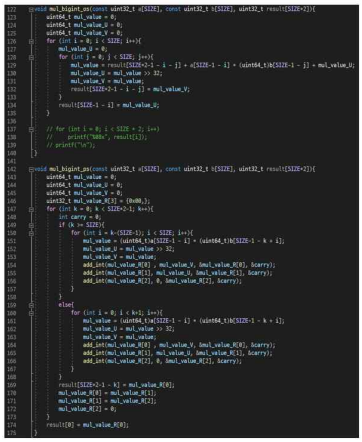
➢ 단돈 25,000원!


```
void mul_schoolBook(bigint* dest, bigint* src1, bigint* src2) {
    uint64_t n = src1->wordlen;
    uint64_t m = src2->wordlen;

    // Temporary bigints for intermediate results
    bigint* temp_wordMUL = NULL;
    bi_new(&temp_wordMUL, 2);
    bigint* temp_add_result = NULL;
    bi_new(&temp_add_result, n + m);

    // Perform multiplication using the schoolbook method
    for (uint64_t j = 0; j < n; j++) {
        for (uint64_t i = 0; i < m; i++) {
            // Initialize temporary bigint to zero
            memset(temp_wordMUL->a, 0, sizeof(word) * temp_wordMUL->wordlen);

            // Multiply each pair of words
            word Aj = src1->a[j];
            word Bi = src2->a[i];
            MULword(temp_wordMUL->a, Aj, Bi);

            // Shift the result and accumulate in the temporary result
            Left_shift(&temp_wordMUL, (i + j) * sizeof(word) * 8);
            add_bigint(&temp_add_result, temp_add_result, temp_wordMUL);
        }
    }

    // Update the sign of the result and assign to the destination bigint
    temp_add_result->sign = src1->sign ^ src2->sign;
    bi_assign(dest, temp_add_result);

    // Cleanup temporary bigints
    bi_delete(&temp_wordMUL);
    bi_delete(&temp_add_result);
}
```

Algorithm 14 Textbook Multiplication

Input: $A = \sum_{j=0}^{n-1} A_j W^j, B = \sum_{j=0}^{m-1} B_j W^j$, where $A_j, B_j \in [0, W)$
Output: $C = AB \in [0, W^{n+m})$

1: **procedure** MULC(A, B)
2: $C \leftarrow 0$
3: **for** $j = 0$ to $n-1$ **do**
4: **for** $i = 0$ to $m-1$ **do**
5: $T \leftarrow A_j B_i$
6: $T \leftarrow T \ll w(i+j)$
7: $C \leftarrow$ ADDC(C, T)
8: **end for**
9: **end for**
10: **return** C
11: **end procedure**



수익모델 2

➢ 추후 ECDSA까지 구현하여 현재 재석이의 AES-GCM 코드와 합쳐 SJS-TLS1.3을 제작!

➢ GPU 포팅하여 GPU 서버에서의 빠르고 많은 양의 데이터 처리 가능한 프로토콜 판매!

➢ GPU를 위한 GCM로직은 현재 특허 출원 진행중임!

➢ 단돈 200만원!