

Computer Security

Term Project_ongoing

- RSA Implement -

학	과	컴퓨터공학과
학	번	201211704
이	름	김기홍
제	출 일	2016.10.02



<목 차>

1 서론	2
2 배경	2
3 구현	3
4 분석	5
5 적용	8
6 회고	9

1 서론

이 보고서는 2016년 2학기 '컴퓨터 보안' 과목의 Term Project에서 파생된 내용이며, 추후에 최종 Term Project 보고서에 포함될 예정이다.

여기서는 RSA 구현에 관련된 내용을 다룬다. 구현을 마치면 Key Generating Time과 RSA의 핵심인 Modulus Operating Time을 개선할 수 있는 대표적인 이진 방식, m진 방식, 소인수 분해 방식, 파워 트리 방식, 가산 고리 방식 중에서 구현하기 가장 쉬운 이진방식을 택하여 속도를 개선시킨다.

속도 측정은 공개키를 3으로 고정시킨 상태에서 $(p-1)*(q-1)$ 값이 3과 서로소가 되는 몇 가지 소수 p, q 에 대하여 key generating time과 Modulus Operating Time의 그래프를 그림으로써 확인 한다.

마지막으로 Term Project의 기본 주제인 'Online Banking' 프로그램의 로그인 과정에 적용할 수 있는 방안에 대해 논의해 보고, 실제로 적용하여 결과를 확인 해보도록 한다.

2 배경

우선 이 프로젝트를 진행한 컴퓨터의 시스템 정보는 다음과 같다.

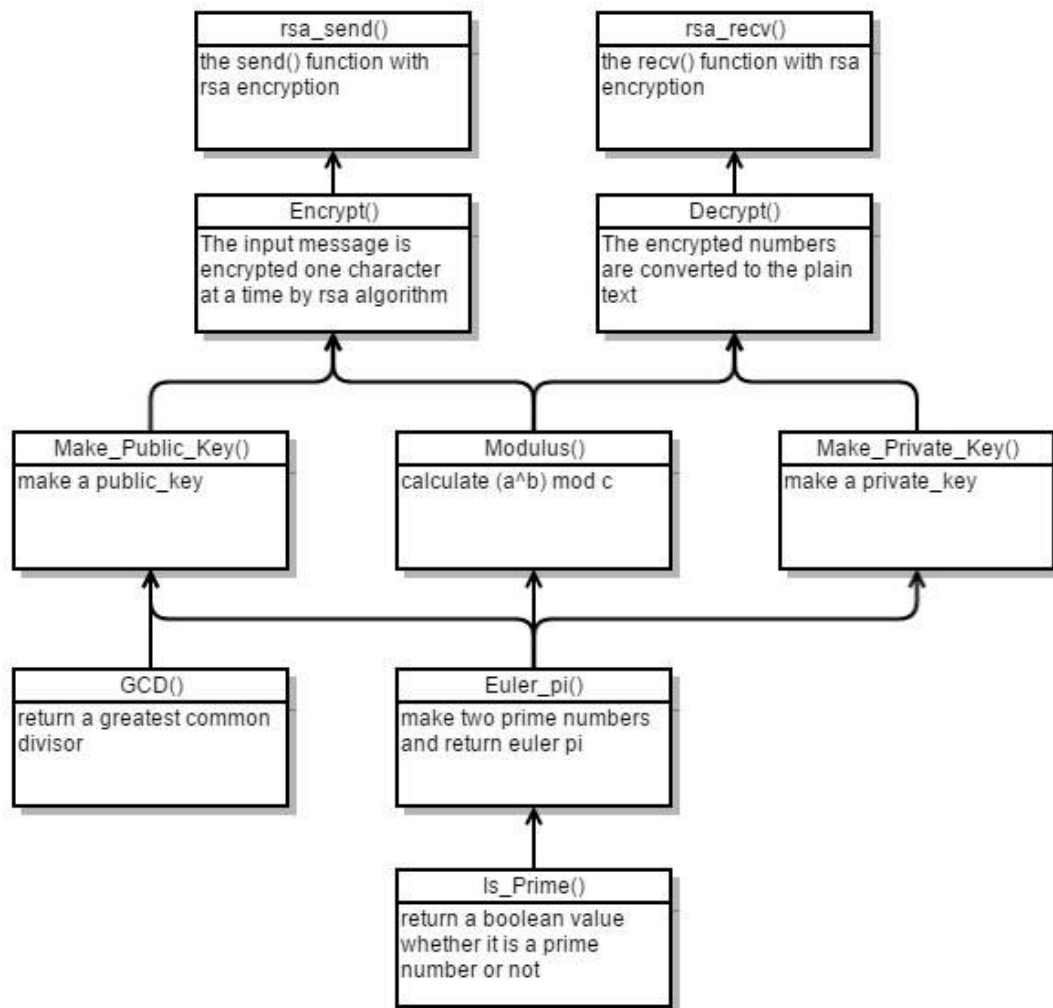
운영체제	Microsoft Windows 10 Pro (64-bit)
프로세서	Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz 2.19 GHz
메모리	8.00GB

개발 툴	Microsoft Visual Studio Community 2015
분석 툴	Wireshark 2.2.0 (64-bit), RawCap

해당 툴들을 선택한 이유는 작성자가 기본적으로 사용하는 툴들이고, 그만큼 조작에 있어서 다른 툴들에 비해 비교적 능숙하기 때문이다.

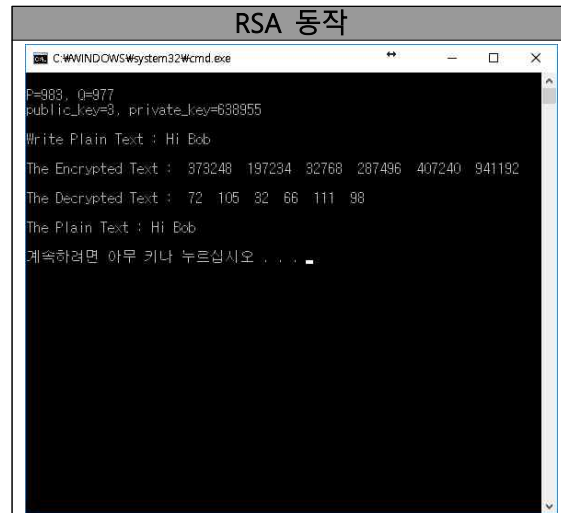
사용한 언어는 C 언어이다.

3 구현



직접 구현한 RSA 라이브러리의 내부 Function Diagram이다. 이번 보고서에서는 동작이 잘 되는지, 그리고 Modulus() 함수와 Make_Private_Key() 함수에 대한 알고리즘 개선 및 시간 측정을 한다.

우선 아래는 시간 측정 없이 RSA 알고리즘이 제대로 동작하는지에 대한 화면을 캡처한 것이다.



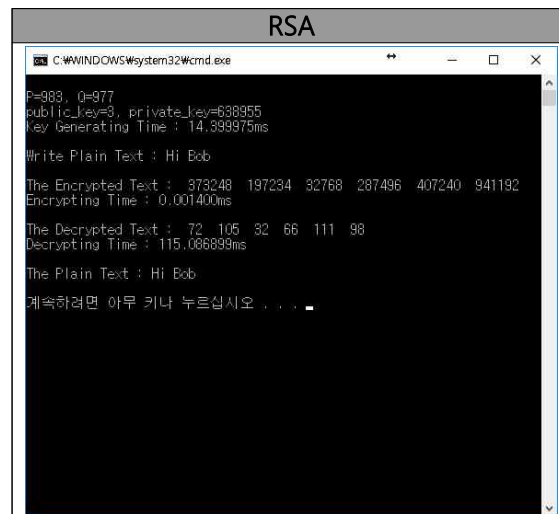
```
C:\WINDOWS\system32\cmd.exe
p=983, q=977
public_key=3, private_key=638955
Write Plain Text : Hi Bob
The Encrypted Text : 373248 197234 32768 287496 407240 941192
The Decrypted Text : 72 105 32 66 111 98
The Plain Text : Hi Bob
계속하려면 아무 키나 누르십시오 . . .
```

이 예제에서 사용된 소수 p 와 q 는 각각 983, 977이며, 공개키는 3을 사용하였다. 그리고 개인키를 계산해내는 연산을 통해 638955를 생성했다.

암호화 방식은 Plain Text의 각 문자에 대한 아스키코드를 RSA 암호화 알고리즘을 사용하여 암호화한다. 복호화는 다시 같은 방법으로 수행함으로써 Plain Text를 얻어낼 수 있다.

4 분석

분석은 공개키 알고리즘에서 중요한 연산 처리 시간을 측정함으로써 진행한다. 아래는 RSA 연산 그대로 구현한 알고리즘이 수행되는 시간을 측정한 것이다.



```
RSA
C:\WINDOWS\system32\cmd.exe
P=983, Q=977
public_key=8, private_key=638955
Key Generating Time : 14.39975ms

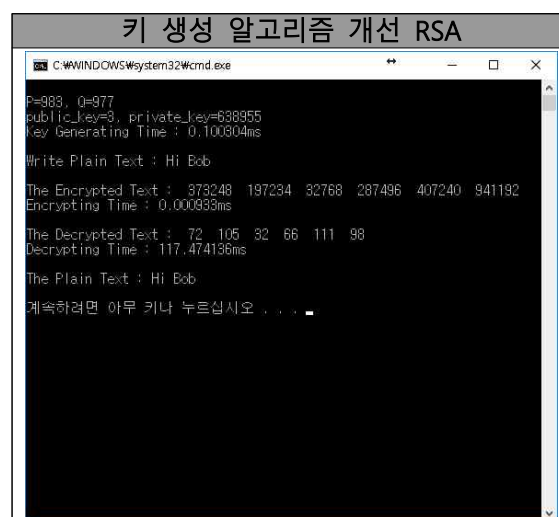
Write Plain Text : Hi Bob

The Encrypted Text : 373248 197234 32768 287496 407240 941192
Encrypting Time : 0.001400ms

The Decrypted Text : 72 105 32 66 111 98
Decrypting Time : 115.086899ms

The Plain Text : Hi Bob
계속하려면 아무 키나 누르십시오 . . .
```

결과를 보면 키를 생성하는 데 약 14.4000ms, 암호화 하는 데 0.0014ms, 복호화 하는 데 무려 115.0869ms나 걸린다. 우선 키를 생성하는 알고리즘을 개선해보자. 기존 개인키를 생성하는 알고리즘은 $ed \bmod (p-1)(q-1) = 1$ 식에서 d 가 조건을 만족할 때 까지 1씩 증가하는 방식이었다. 이 알고리즘은 위의 식을 약간 변형하여 개선시킬 수 있다. $d = ((p-1)(q-1)x + 1)/e$ 를 만족하는 정수 d 를 찾는 방식을 사용하면 시간을 단축시킬 수 있다. 아래는 키 생성 알고리즘을 개선한 후 측정한 시간이다.



```
키 생성 알고리즘 개선 RSA
C:\WINDOWS\system32\cmd.exe
P=983, Q=977
public_key=8, private_key=638955
Key Generating Time : 0.100304ms

Write Plain Text : Hi Bob

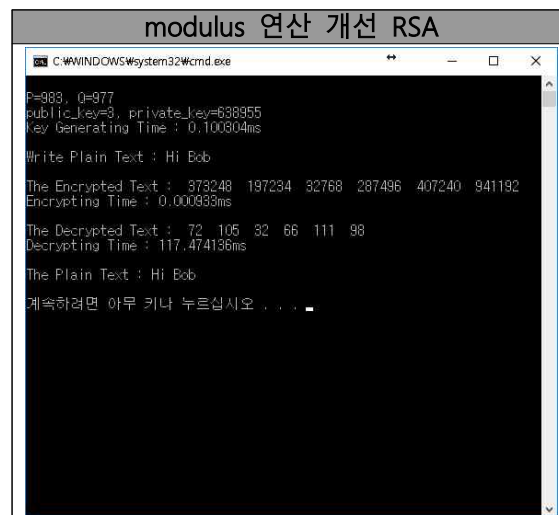
The Encrypted Text : 373248 197234 32768 287496 407240 941192
Encrypting Time : 0.000933ms

The Decrypted Text : 72 105 32 66 111 98
Decrypting Time : 117.474136ms

The Plain Text : Hi Bob
계속하려면 아무 키나 누르십시오 . . .
```

키를 생성 하는 데 소요된 시간이 14.4000ms에서 0.100304ms로 단축된 것을 볼 수 있다.

다음은 암호화, 복호화 하는 데 큰 영향을 미치는 Modulus 연산 알고리즘을 개선시켜보자. 방식은 이진 방식을 사용하였다. 이진 방식이란 간단하게 설명하면 $a^b \bmod c$ 식을 계산하고자 할 때 b를 이진수로 나타내어 최상위 비트부터 비트가 1이면 a를 곱하고 0이면 제곱을 한다. 이 방법을 사용하면 곱셈 연산의 횟수도 급격히 줄어들 뿐더러 계산 도중 큰 수가 나오지 않기 때문에 빠른 연산이 가능하다.



```
modulus 연산 개선 RSA
C:\WINDOWS\system32\cmd.exe
P=983, Q=977
public_key=3, private_key=638955
Key Generating Time : 0.100304ms

Write Plain Text : Hi Bob

The Encrypted Text : 373248 197234 32768 287496 407240 941192
Encrypting Time : 0.000933ms

The Decrypted Text : 72 105 32 66 111 98
Decrypting Time : 117.474136ms

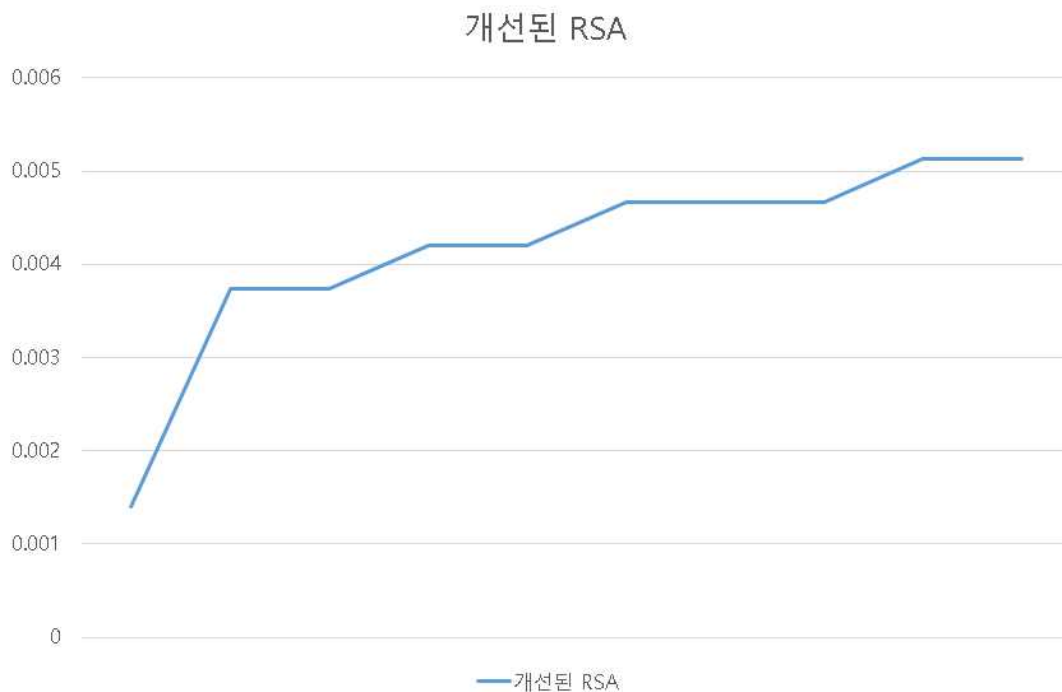
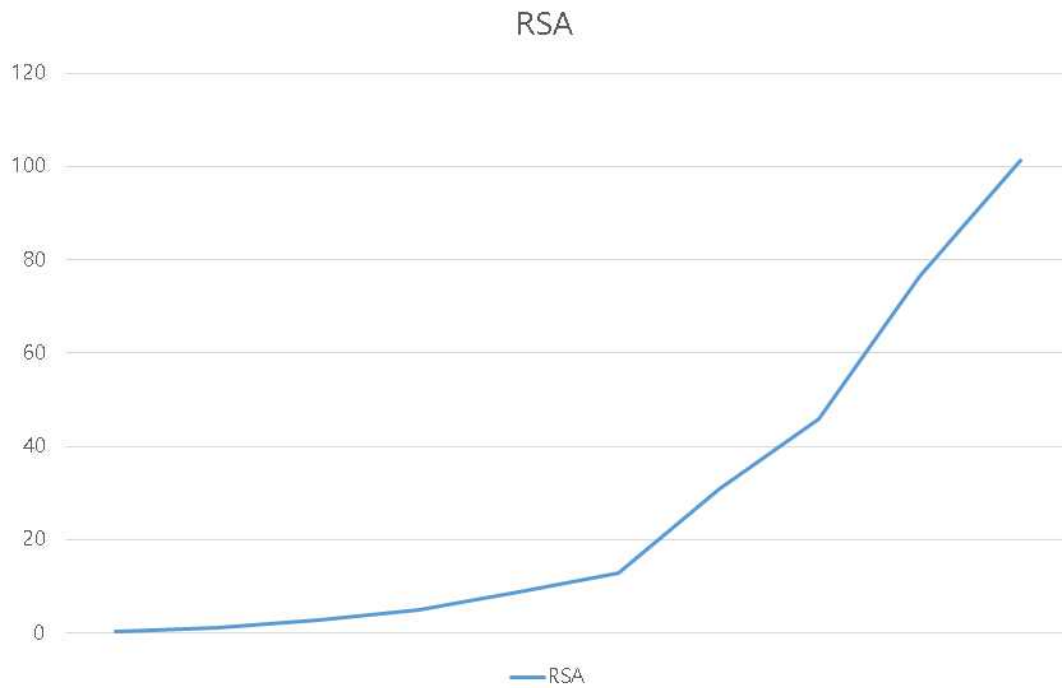
The Plain Text : Hi Bob

계속하려면 아무 키나 누르십시오 . . .
```

복호화 하는 데 소요된 시간이 115.0869ms에서 0.0056ms로 단축되었다. 대략 50000배 정도 감소된 것을 확인할 수 있다.

마지막으로 소수의 크기에 따른 복호화 시간을 개선 전 알고리즘, 그리고 개선 후 알고리즘을 대상으로 하여 그래프로 나타내어 보자.

사용된 소수들의 <p,q>순서쌍은 <41, 47> <89,101> <149,167> <191,227> <251,269> <311,317> <491,509> <617,647> <761,797> <887,929>이다. 그리고 사용된 메시지는 "Hi Bob"이다. y축은 ms 단위이다.



그래프를 보면 양쪽 모두 소수가 증가함에 따라 시간이 증가하는 것을 볼 수 있다. 결론은 RSA 알고리즘을 안전하게 사용하기 위해서는 소수의 크기가 커야 하는데, 그에 따라 소요 시간도 많이 소요되기 때문에 연산 처리를 빠르게 하는 알고리즘이 중요시 된다.

5 적용

'Online Banking' 프로그램의 로그인 과정에서 이번에 구현한 RSA를 사용하기 위해서는 우선 서버와 클라이언트 양측 사이에 각자의 공개키를 공유하고 있다는 가정이 필요하다. 공개키를 PKI를 통해 조금 더 효율적으로 관리할 수도 있겠지만 여기서는 각자의 공개키를 그냥 주고받기로 한다.

각 프로그램은 메시지를 전송하기 전에 자신의 개인키로 메시지를 암호화하여 전송하고, 수신 받은 메시지를 송신자의 공개키를 이용하여 복호화 해야 한다. 여기서 메시지는 각 문자별로 아스키코드를 사용해 암호화 및 복호화 되는데, 이 때 RSA는 숫자를 사용하기 때문에 각 문자를 구별할 수 있도록 어떤 토큰이 필요하다. 필자는 각 문자의 암호화된 숫자 사이에 '@' 문자를 삽입함으로써 문자들을 구별할 수 있었다.

H i _ B o b
~@~@~@~@~@~@~@~@~@

'~'는 각 문자에 해당하는 암호화된 숫자를 의미한다. 위와 같은 모양으로 메시지를 전송하게 되면, 수신측에서는 '@'를 토큰삼아 얻어진 각각의 숫자들을 자신의 개인키로 복호화하게 되면 본래의 메시지를 얻을 수 있다. 실제 사례 적용 후 WireShark를 통해 분석한 결과를 한 번 살펴보자.

ID_PW Sniffing			
40	9000→55467	[ACK]	Seq=307957336 Ack=4 Win=525568 Len=0
50	55467→9000	[PSH, ACK]	Seq=4 Ack=307957336 Win=525568 Len=10
18	08 05 96 b1 00 00	72 6b 66 61 6f 72 6c 30	P..... rkfaorl0
30			00
40	9000→55467	[ACK]	Seq=307957336 Ack=14 Win=525568 Len=0
49	55467→9000	[PSH, ACK]	Seq=14 Ack=307957336 Win=525568 Len=9
18	08 05 94 d7 00 00	72 6b 66 61 6f 72 6c 31	P..... rkfaorl1
32			2

ID_PW Sniffing_RSA										
40 9000→55777 [ACK] Seq=117 Ack=235 Win=525312 Len=0										
99 55777→9000 [PSH, ACK] Seq=235 Ack=117 Win=525312 Len=59										
04	70	34	00	00	32	32	36	35	32 40 32 30	P...p4.. 22652@20
40	35	34	36	34	32	40	33	31	38 32 40 35	793@5464 2@3182@5
33	40	32	32	36	35	32	40	33	30 33 32 38	7613@226 52@30328
30	35	32	40	35	38	30	35	32	40 35 38 30	@58052@5 8052@580
										52@
40 9000→55777 [ACK] Seq=117 Ack=294 Win=525056 Len=0										
93 55777→9000 [PSH, ACK] Seq=294 Ack=117 Win=525312 Len=53										
04	20	85	00	00	32	32	36	35	32 40 32 30	P... .. 22652@20
40	35	34	36	34	32	40	33	31	38 32 40 35	793@5464 2@3182@5
33	40	32	32	36	35	32	40	33	30 33 32 38	7613@226 52@30328
32	37	39	40	31	36	37	32	39	40	@48279@1 6729@

6 회고

RSA를 구현하면서 어려웠던 점은 수학적으로 이루어진 알고리즘을 컴퓨터에 맞게 구현하는 것, 그리고 계산 속도를 더욱 향상시키는 방법을 구상해내는 것이었다. 특히 계산 속도를 향상시키는 방법에는 여러 가지가 있었지만 안타깝게도 필자 혼자 이해하기가 너무 어려웠고 그 중 가장 간단하고 이해하기 쉬운 이진 방식을 택할 수 없던 점이 몹시 아쉽다. 처음에는 다른 방법들과도 속도를 비교해보고 싶은 마음이었지만 그렇게 하지 못해 조금 마음이 불편하다. 하지만 아쉬운 점이 많은 만큼 배운 점도 많다. 비록 수학적인 내용들이 대부분이어서 크게 실용적이진 못했지만 새로운 것을 알게 되어 뿌듯하다.