

# Computer Security

## Term Project

- Programming a secure online bank -

학	과	컴퓨터공학과
학	번	201211704
이	름	김기홍
제	출 일	2016.11.06



## <목 차>

1 서론 .....	4
2 배경 .....	5
3 시스템 구조 및 시연 .....	6
4 암호화 모듈 .....	10
4.1 A5/1 .....	10
4.2 Feistel 구조 기반 Block 암호 .....	12
4.3 RSA 암호 .....	13
4.4 ECC 디피-헬먼 키 교환 .....	15
4.5 SHA-256 .....	17
5 MiM(Man in the Middle) Attack Test .....	18
6 회고 .....	22
7 부록 .....	25

## <그림 목차>

그림 2.1 프로젝트 진행 컴퓨터 시스템 정보 .....	5
그림 2.2 사용된 툴들 .....	5
그림 3.1 전체 시스템 구조 .....	6
그림 3.2 프로그램 실행 시 화면 .....	6
그림 3.2 계정을 등록하는 화면 .....	7
그림 3.3 각 프로그램이 생성한 파일_1 .....	7
그림 3.4 로그인하는 화면 .....	7
그림 3.5 각 프로그램이 생성한 파일_2 .....	8
그림 3.6 계좌 조회, 입금, 출금 화면 및 Database.txt 내용 변화 .....	8
그림 3.7 다양한 오류 화면 .....	9
그림 4.1 암호화 모듈 라이브러리 .....	10
그림 4.1.1 A5/1 암호화 모듈 사용법 .....	10
그림 4.1.2 A5/1 키스트림 구조 .....	11
그림 4.1.3 A5/1 암호화 시연 화면 .....	11
그림 4.2.1 Feistel 구조 블록 암호화 모듈 사용법 .....	12
그림 4.2.2 Feistel 구조 블록 암호화 알고리즘 .....	12
그림 4.2.3 Feistel 구조 블록 암호화 시연 화면 .....	13
그림 4.3.1 RSA 암호화 모듈 사용법 .....	13
그림 4.3.2 RSA 암호화 시연 화면 .....	14
그림 4.4.1 ECC 디피-헬먼 키 교환 모듈 사용법 .....	15
그림 4.4.2 서버 측 ECC 디피-헬먼 키 교환 시연 화면 .....	16
그림 4.4.3 클라이언트 측 ECC 디피-헬먼 키 교환 시연 화면 .....	16
그림 4.5.1 SHA-256 시연 화면 .....	17
그림 4.5.2 SHA-256 시연 화면(Database.txt) .....	17
그림 5.1 Man in the Middle Attack Diagram .....	18
그림 5.2 Man in the Middle Attack Simple Diagram .....	19
그림 5.3 Bob의 로그인 화면(Trudy가 훔쳐봄) .....	19
그림 5.4 WireShark를 통한 Bob의 로그인 패킷 분석 .....	20
그림 5.5 트루디의 Cut and Paste Attack 화면 .....	20
그림 5.6 WireShark를 통한 트루디의 Cut and Paste Attack 패킷 분석 .....	21
그림 6.1 서버 측 사용자 인증의 필요성 .....	22
그림 6.2 신뢰할 수 없는 서버 .....	23
그림 6.3 NONCE 기법 .....	23
그림 6.4 Hashed NONCE 기법 .....	24

## 1 서론

본 프로젝트는 '온라인 뱅킹 시스템'을 기본 모델로 인증, 인허, 암호, 보안 프로토콜 등의 보안 시스템을 이해하고 개발하는 것이 목적이다. 이전 프로젝트에서는 은행 서버와 고객 클라이언트로 구성된 2-Tier 구조를 사용했다. 구현했던 기능은 계정 등록, 인증 후 계좌 조회, 입금, 출금 기능과 RSA 알고리즘을 통한 암호화 메시지 전송 기능이였다. 공개키는 각 프로그램 실행 시 공개키, 개인키를 생성한 후 공개키를 서로 교환함으로써 배포하였고 서버는 멀티스레드 방식으로 여러 클라이언트를 동시에 접속시킬 수 있었다. 각 트랜잭션은 모두 파일로 기록되며 인증 정보(ID, PW)는 서버에 파일로 저장되는 방식이었다. 이 시스템의 문제로는 크게 메시지 무결성 문제, 서버 측 사용자 정보의 보안 미흡 등이 있었다.

이번 프로젝트에서는 외부에서 중간자가 데이터를 훔쳐보거나 변조가 가능하도록 2-Tier 구조를 3-Tier 구조로 변경했다. 이 과정에서 RSA 암호화를 포함한 스트림 암호 A5/1, Feistel 구조를 사용하여 만든 별도의 블록 암호 모듈을 추가했다. 또한 대칭키 암호의 문제점인 키 교환 문제를 개선하기 위해 ECC 디피-헬먼 방식을 적용한 'Key\_Gender' 모듈을 직접 구현하여 사용했다. 이 외에도 서버 측 인증 정보가 노출될 가능성을 고려하여 서버 측 사용자 인증 정보를 암호화했다. 특히 ID, PW는 KISA에서 제공하는 SHA-256 라이브러리를 통해 해시 과정을 거친 후 보관하도록 구현했다. 또한 TCP 프로토콜의 메시지 경계 문제를 고정 길이 메시지를 주고받음으로써 해결하였다.

이전보다 개선시킨 시스템임에도 불구하고 아직 여러 보안적 문제점들이 존재한다. 그 문제점들을 간단하게 설명하자면 우선 현 시스템은 사용자는 인증하고 있지만 더욱 중요시 되어야 할 서비스를 제공하는 서버를 인증하고 있지 않다는 점이다. 또 현 시스템은 서버가 사용자의 패스워드를 수신 받아서 해시하기 때문에 사용자들의 패스워드를 알고 있고, 패스워드가 알려지는 것은 결코 바람직하지 못한 상황이다. 중간자 공격(Cut and Paste Attack) 또한 여전히 유효하며 이는 5장에서 실제 테스트 해본다. 그리고 아직까지 적용되지 않은 보안 프로토콜과 소프트웨어 측면의 위험성 배제 또한 현 시스템의 풀어야 할 과제이다.

이 보고서에서는 먼저 프로젝트를 진행한 시스템의 사양을 소개하고 이번 프로젝트의 시스템 구조, 암호화 모듈의 구현 및 동작, 중간 서버에 침입하여 데이터를 변조하는 MiM(Man in the Middle) 공격, 그리고 트루디의 공격법 및 대응 방안에 대해 설명하고 진행 과정에서의 문제와 해결 및 배운 점을 기술한다. 마지막 7장 부록에서는 고전 암호 중 카이사르 암호를 소개 및 시연하고 카이사르 암호문을 적용시킨 에드거 앨런 포의 소설 『The Gold Bug』의 알파벳 빈도수를 조사하여 키 전수 검사 없이 한 번에 Breaking하는 프로그램을 소개한다.

## 2 배경

우선 이 프로젝트를 진행한 컴퓨터의 시스템 정보는 다음과 같다.

구분	내용
운영 체제	Microsoft Windows 10 Pro (64-bit)
프로세서	Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40 GHz
메모리	8.00GB

그림 2.1 프로젝트 환경

그리고 다음은 사용된 툴들의 정보이다.

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

그림 2.2 사용한 툴

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

‘온라인 뱅킹’ 프로그램의 구현은 TCP 연결 기반의 서버, 릴레이 서버, 클라이언트로 이루어지고, 서버와 릴레이 서버는 멀티 스레드 환경을 지원하기 때문에 여러 클라이언트가 동시에 접속할 수 있다. 릴레이 서버의 포트 번호는 9000번, 서버의 포트 번호는 9001번을 사용하였다.

### 3 시스템 구조 및 기능 시연

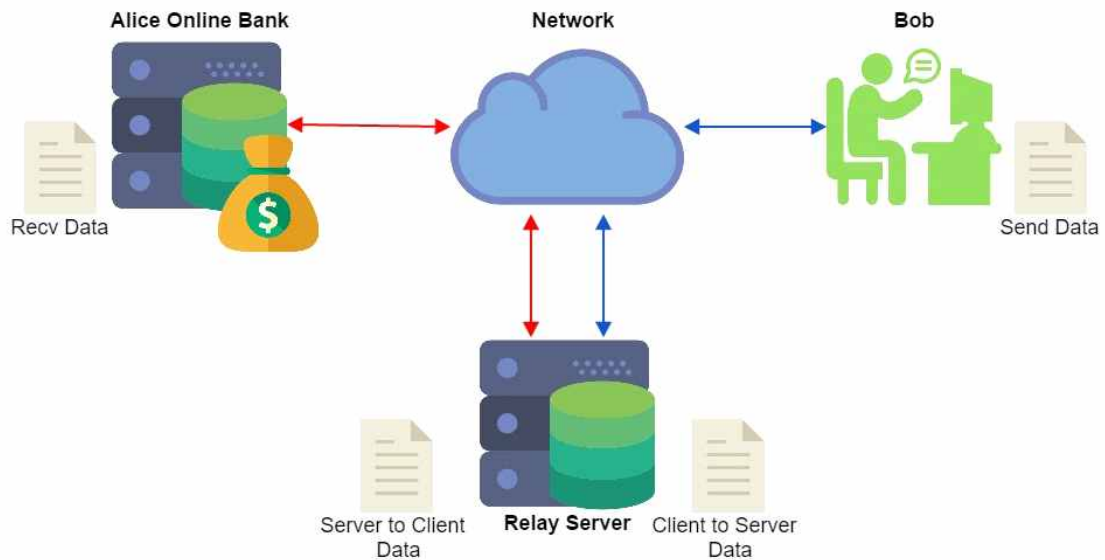


그림 3.1 전체 시스템 구조

그림 3.1은 현재 시스템의 전체적인 모습을 보여주는 다이어그램이다. 최초 클라이언트 프로그램 실행 시 릴레이 서버의 한 스레드로 접속하고, 그 스레드는 다시 서버의 한 스레드로 접속한다. 따라서 모든 메시지는 릴레이 서버를 거친다.

서버	<pre> ca. Server [ TCP 서버 ] 클라이언트 접속 : IP주소=127.0.0.1, 포트번호=58418 </pre>
릴레이 서버	<pre> ca. Relay Server [ TCP 서버 ] 클라이언트 접속 : IP주소=127.0.0.1, 포트번호=58417 _ </pre>
클라이언트	<pre> ca. Client 1. 로그인 2. 비밀번호 3. 계좌정보 기능 선택: _ </pre>

그림 3.2 프로그램 실행 시 화면

또, 각 프로그램은 각자 파일 입출력을 통해 어떤 정보를 관리한다. 예를 들어, 클라이언트는 수행한 트랜잭션의 종류 및 전송한 정보를 로그 파일 형식으로 저장하고, 릴레이 서버는 클라이언트에서 서버로, 또 서버에서 클라이언트로 전송되는 정보들을 저장한다. 마지막으로 서버는 사용자 정보 즉, ID와 PW, 이름, 잔액을 저장한다.

서버	
릴레이 서버	
클라이언트	

그림 3.2 계정을 등록하는 화면

서버	
릴레이 서버	
클라이언트	

그림 3.3 각 프로그램이 생성한 파일\_1

릴레이 서버는 클라이언트가 전송한 데이터를 그림 3.3에서 보이는 바와 같이 파일로 저장한다. 그리고 다시 그 파일을 읽어서 서버에게 전송한다. 이는 트루디가 전송되는 내용을 변조할 수 있도록 하기 위함이다.

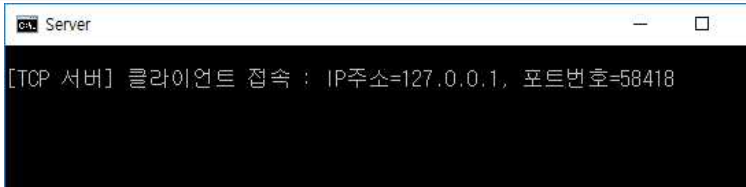
서버	
릴레이 서버	
클라이언트	

그림 3.4 로그인하는 화면

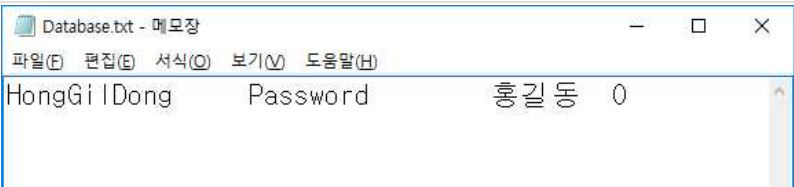
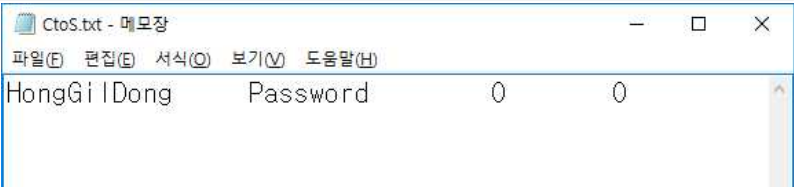
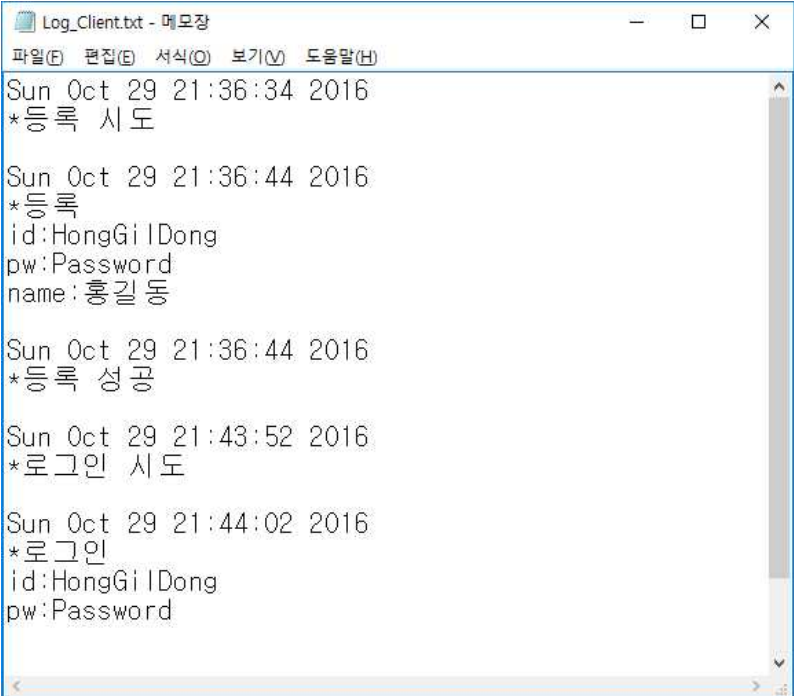
서버	
릴레이 서버	
클라이언트	

그림 3.5 각 프로그램이 생성한 파일\_2

서버는 실행 시, 자신이 보관하고 있는 Database.txt 파일을 읽어서 그 정보들을 연결리스트에 보관한다. 실행 중 클라이언트가 계정을 등록할 경우 아이디를 비교하여 중복이 없다면 연결리스트에 추가하고, 그 정보를 파일에도 쓴다. 클라이언트가 로그인을 시도할 경우 전송받은 정보와 연결리스트의 정보를 비교하여 인증 과정을 수행한다.



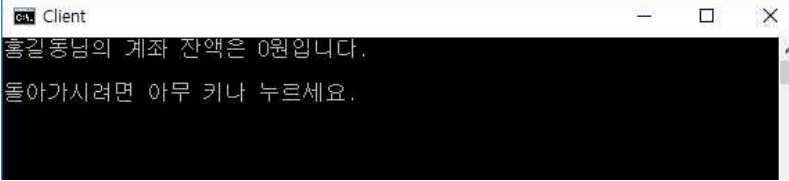
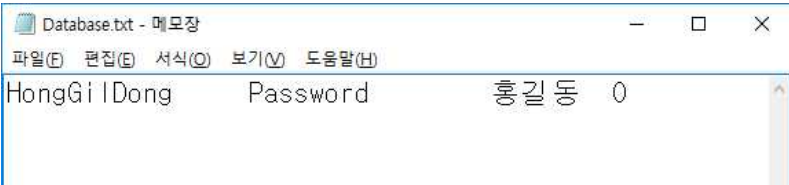
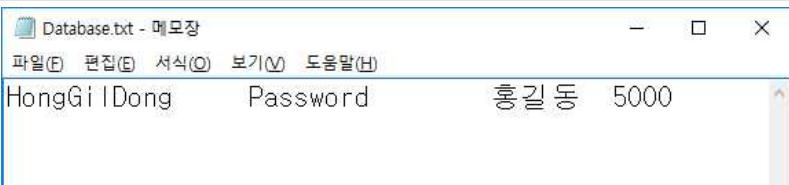
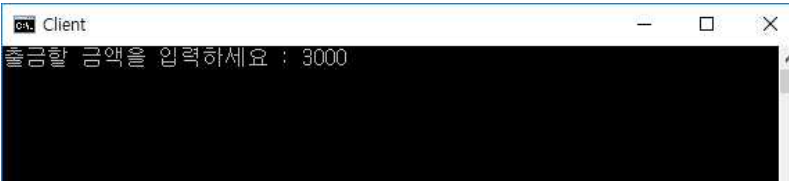
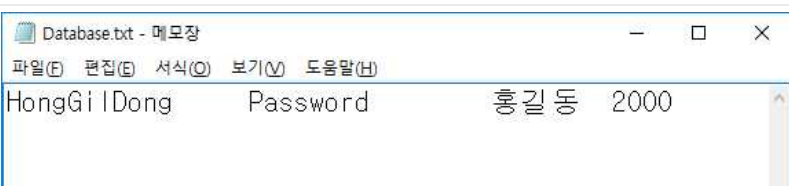
계좌 조회	 
입금	 
출금	 

그림 3.6 계좌 조회, 입금, 출금 화면 및 Database.txt 내용 변화

계좌 조회, 입금, 출금 기능이 수행될 때마다 서버는 전송받은 값이 타당한지 검사한 후에 자신의 연결리스트 및 Database.txt를 갱신하고, 클라이언트에게 이름 및 잔액을 전송한다.

정형적인 기능 소개는 마치고, 조금 다른 경우에 따른 다양한 화면들을 한번 살펴보고 이번 장을 마치도록 한다.

중복된 아이디 등록	
아이디 불일치	
비밀번호 불일치	
잘못된 금액 입금	
잘못된 금액 출금	
잘못된 메뉴 선택	

그림 3.7 다양한 오류 화면

## 4 암호화 모듈



그림 4.1 암호화 모듈 라이브러리

암호화 모듈들은 서버, 릴레이 서버, 클라이언트 프로그램에 공통으로 사용되는 모듈들과 함께 하나의 라이브러리로 구성되어진다. 이번 장에서는 구체적인 구현 설명은 뒤로하고, 사용하는 방법과 실제 동작 화면을 보인다.

### 4.1 A5/1

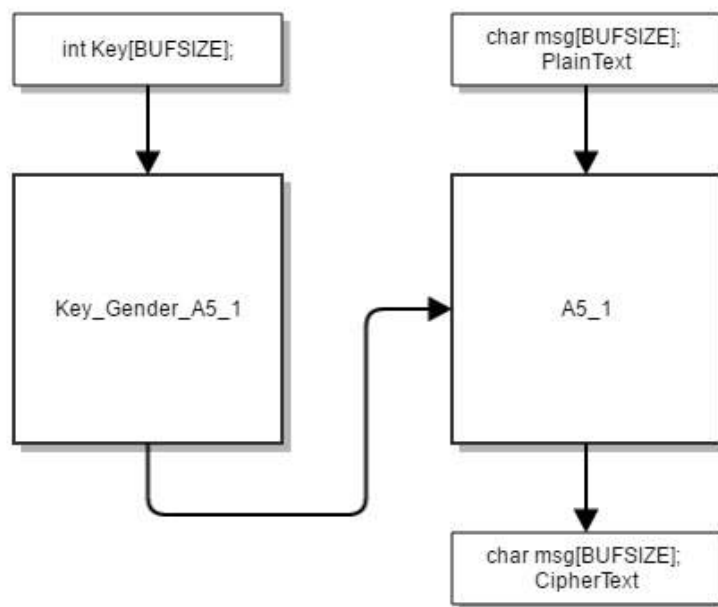


그림 4.1.1 A5/1 암호화 모듈 사용법

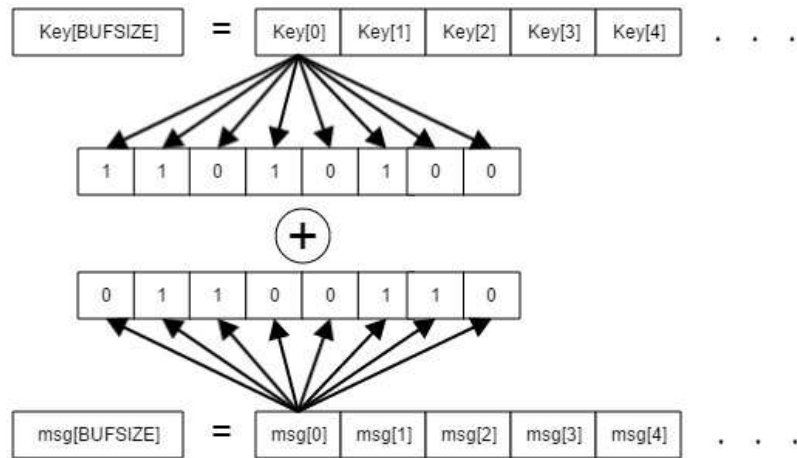


그림 4.1.2 A5/1 키스트림 구조

A5/1 암호화는 그림 4.1.1처럼 정수형 배열을 선언한 후, `Key_Gender_A5_1`함수를 사용해 각 인덱스 하나 당 8비트짜리 키스트림을 연달아 생성한다. 암호화는 문자열의 각 문자 하나당 이루어진다. (8비트짜리 키스트림을 저장하는 배열은 `int`형이 아닌 `short`형으로 구현하는 것이 메모리 효율에 좋지만 작성자 판단에 여러 암호화 모듈에서 사용할 수 있도록 정수형을 선택해 작성하였다.)

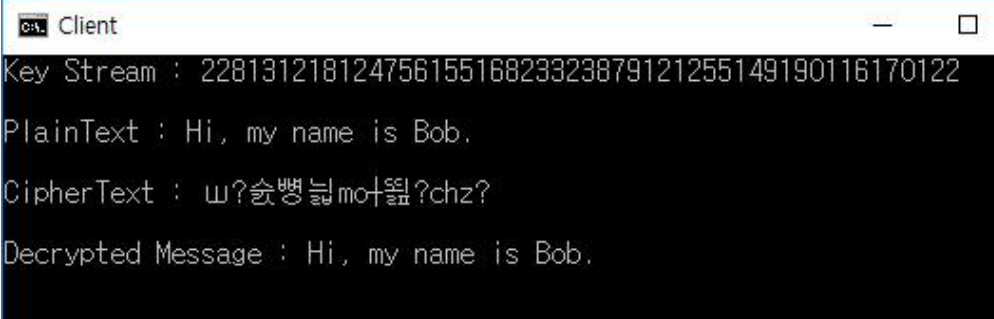
A5/1 암호화 시연	 <pre> Client Key Stream : 22813121812475615516823323879121255149190116170122 PlainText : Hi, my name is Bob. CipherText : ㅍ?슌뵙늬모아뵙?chz? Decrypted Message : Hi, my name is Bob. </pre>
A5/1 암호화 시연 코드	<pre> int Key[BUFSIZE];  Key_Gender_A5_1(Key); printf("Key Stream : "); for (int i = 0; i &lt; BUFSIZE; i++)     printf("%d", Key[i]); puts(""); puts(""); strcpy(msg, "Hi, my name is Bob."); // 평문 printf("PlainText : %s WnWn", msg); A5_1(msg, Key);                      // 암호화 printf("CipherText : %s WnWn", msg); A5_1(msg, Key);                      // 복호화 printf("Decrypted Message : %s WnWn", msg); </pre>

그림 4.1.3 A5/1 암호화 시연 화면

## 4.2 Feistel 구조 기반 Block 암호

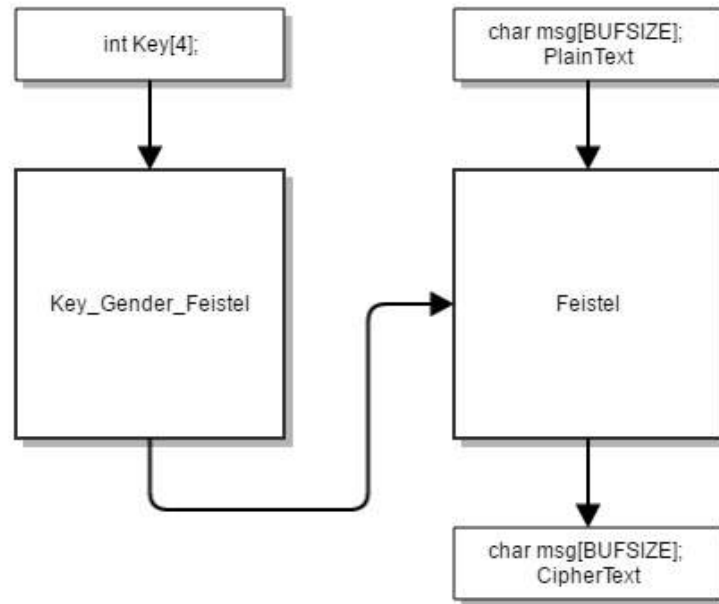


그림 4.2.1 Feistel 구조 블록 암호화 모듈 사용법

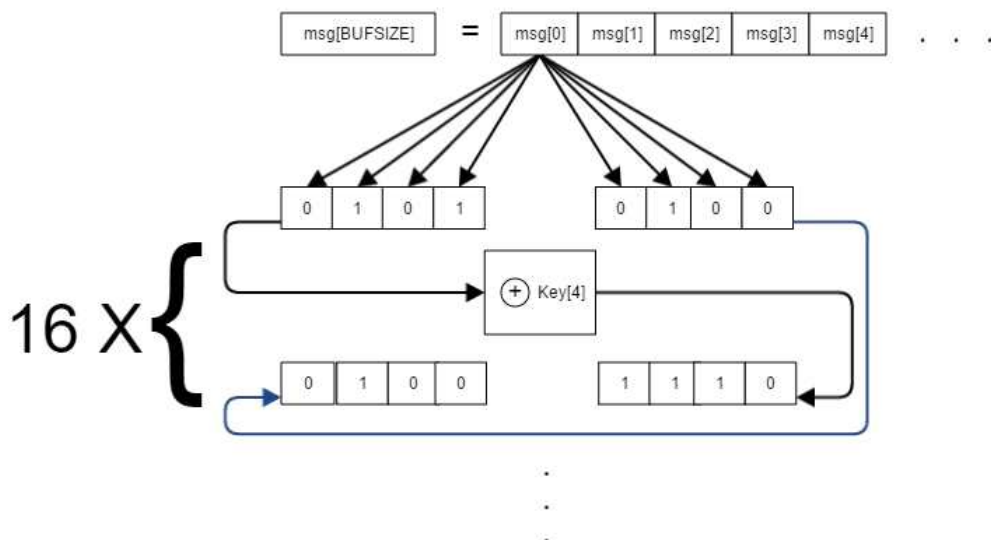


그림 4.2.2 Feistel 구조 블록 암호화 알고리즘

Feistel 구조 암호화는 정말 단순한 알고리즘을 사용하였다. 그 이유는 우선 블록 암호의 구조 중 하나인 Feistel 구조의 이해와 구현에 우선 중점을 두었기 때문이다. 문자 하나를 하나의 블록으로 정하여, 좌우 4비트씩 나눈다. 그리고 오른쪽 블록은 그대로 다음 왼쪽 블록으로, 왼쪽 블록은 4비트 키와 XOR연산을 한 후 오른쪽 블록으로 보낸다. 이 과정을 16번 반복하는 알고리즘이다. 추후에는 블록 암호의 모드를 구현 함으로써 메시지 무결성을 보장하는 데에도 기여를 할 것으로 예상된다. (키가 4비트이기 때문에 보안적인 측면에서는 정말 극단적으로 취약하지만, 오히려 구조를 이해하고 직접 구현하는 데는 큰 도움이 되었다.)

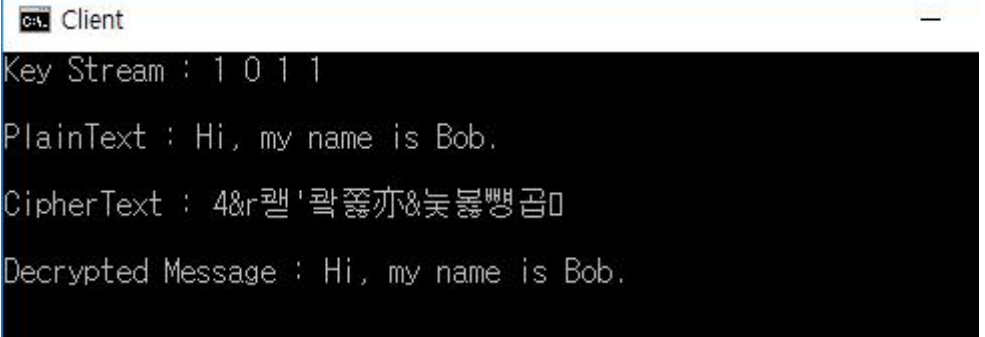
<p>Feistel 구조 블록 암호화 시연</p>	 <pre> c++. Client Key Stream : 1 0 1 1 PlainText : Hi, my name is Bob. CipherText : 4&amp;r광'광쵸亦&amp;늣뵡뵡곶 Decrypted Message : Hi, my name is Bob. </pre>
<p>Feistel 구조 블록 암호화 시연 코드</p>	<pre> int Key[BUFSIZE];  Key_Gender_Feistel(Key); printf("Key Stream : "); for (int i = 0; i &lt; 4; i++)     printf("%d ", Key[i]); puts(""); puts(""); strcpy(msg, "Hi, my name is Bob."); // 평문 printf("PlainText : %s \n\n", msg); Feistel(msg, Key, ENC);              // 암호화 printf("CipherText : %s \n\n", msg); Feistel(msg, Key, DEC);              // 암호화 printf("Decrypted Message : %s \n\n", msg); </pre>

그림 4.2.3 Feistel 구조 블록 암호화 시연 화면

### 4.3 RSA 암호

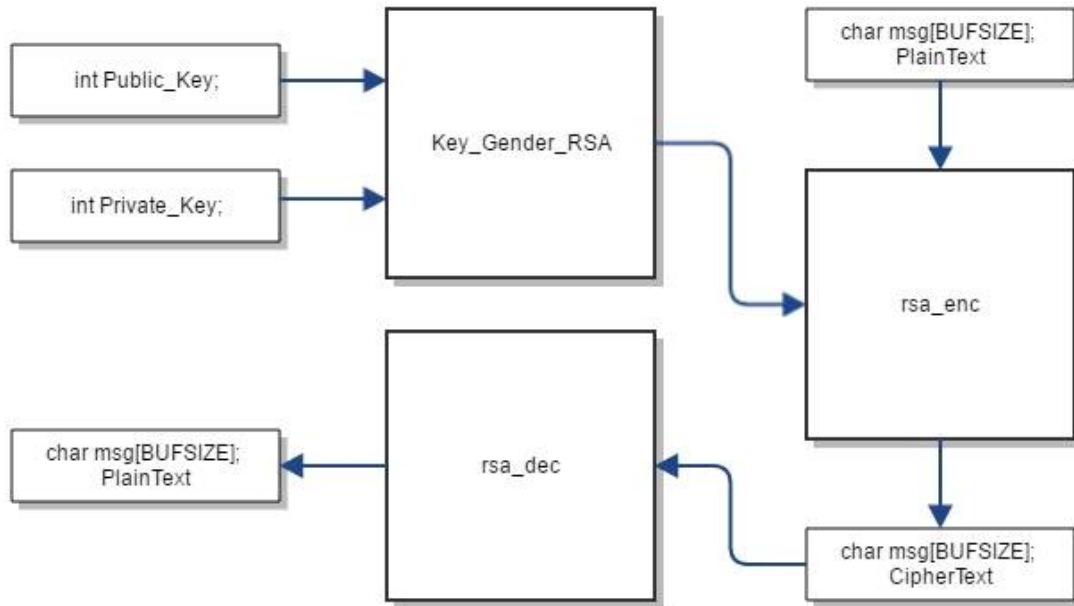


그림 4.3.1 RSA 암호화 모듈 사용법

난수로 만들어진 소수  $p$ ,  $q$ 와 그 소수들로 구한  $e_{pi}$  값을 사용해 공개키와 개인키를 `Key_Gender_RSA` 모듈로 구하고, 그 키들을 사용하여 `rsa_enc`, `rsa_dec` 모듈로 문자열을 암호화 및 복호화 할 수 있다.



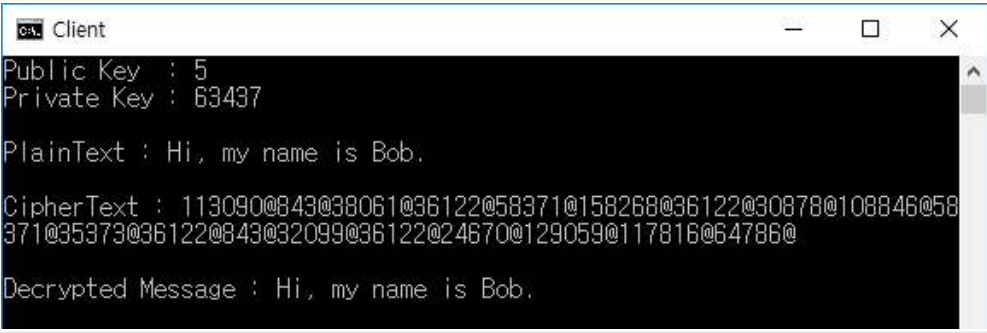
RSA 암호화 시연	 <pre> Client Public Key : 5 Private Key : 63437  PlainText : Hi, my name is Bob.  CipherText : 113090@843@38061@36122@58371@158268@36122@30878@108846@58 371@35373@36122@843@32099@36122@24670@129059@117816@64786@  Decrypted Message : Hi, my name is Bob. </pre>
RSA 암호화 시연 코드	<pre> int public_key, private_key;  Key_Gender_RSA(&amp;public_key, &amp;private_key); printf("Public Key : %d\n", public_key); printf("Private Key : %d\n", private_key); puts(""); strcpy(msg, "Hi, my name is Bob."); // 평문 printf("PlainText : %s\n\n", msg); rsa_enc(msg, public_key);           // 암호화 printf("CipherText : %s\n\n", msg); rsa_dec(msg, private_key);          // 암호화 printf("Decrypted Message : %s\n\n", msg); </pre>

그림 4.3.2 RSA 암호화 시연 화면

여기서 메시지는 각 문자별로 아스키코드를 사용해 암호화 및 복호화 되는데, 이 때 RSA는 숫자를 사용하기 때문에 각 문자를 구별할 수 있도록 어떤 토큰이 필요하다. 작성자는 각 문자의 암호화된 숫자 사이에 '@' 문자를 삽입함으로써 문자들을 구별할 수 있었다. 예를 들면 다음과 같다.

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.

#### 4.4 ECC 디피-헬먼 키 교환

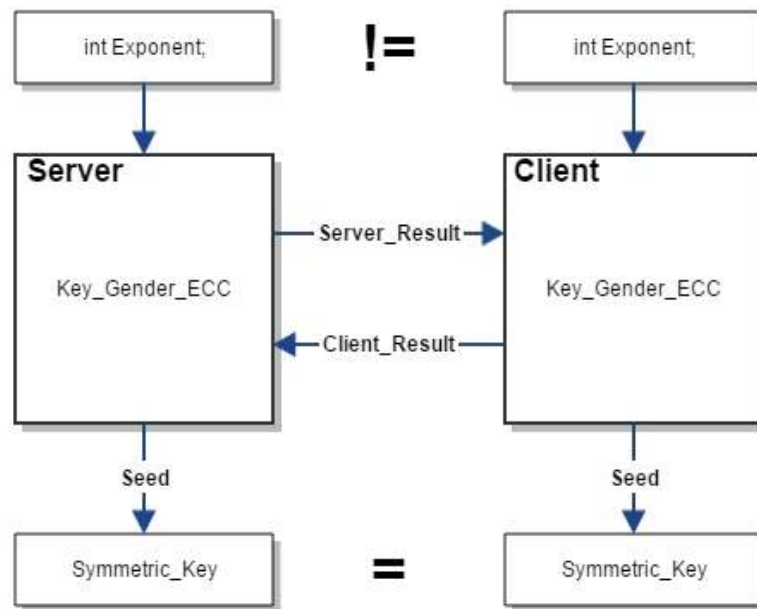


그림 4.4.1 ECC 디피-헬먼 키 교환 모듈 사용법

ECC 디피-헬먼 키 교환 모듈은 각자의 비밀 승수를 모듈에 매개변수로 넣으면 자동적으로 내부에서 ECC 연산이 이루어지고 통신이 이루어진다. 그 후 다시 한 번 더 ECC 연산이 이루어지면 양쪽이 타원 곡선 위의 같은 한 점을 얻게 되는데, 그 점을 Seed로 사용하여 동일한 Symmetric\_Key를 생성한다. ECC 연산을 위한 타원 곡선 방정식 및 나머지 연산 값과 개인의 승수 값은 랜덤하게 생성하지만 이번 시연에서는 우선 교과서에 나오는 값으로 설정 해놓았다. 그 이유는 교과서에 나오는 값을 사용함으로써 제대로 된 연산이 이루어지는 것을 증명하기 위해서이다.

<p>서버 ECC 디피-헬먼 키 교환 시연</p>	 <pre> [TOP 서버] 클라이언트 접속 : IP주소=127.0.0.1, 포트번호=51896 <math>y^2 = x^3 + 11x + 19 \pmod{167}</math> , P(2,7)  My Private Exponent is 22 22 * (2, 7) = (9, 43) 22 * (102, 88) = (131, 140) Symmetric_Key : 000100000111100101111000111110 </pre>
<p>서버 ECC 디피-헬먼 키 교환 시연 코드</p>	<pre> Thread_Proc(void* sock)  SOCKET client_sock = (SOCKET)sock;  int Bob_Exponent = 22; int Symmetric_Key[BUFSIZE];  Key_Gender_ECC(client_sock, Symmetric_Key, Bob_Exponent);  printf("Symmetric_Key : "); for (int i = 0; i &lt; BUFSIZE; i++)     printf("%d", Symmetric_Key[i]); puts(""); </pre>

그림 4.4.2 서버 측 ECC 디피-헬먼 키 교환 시연 화면

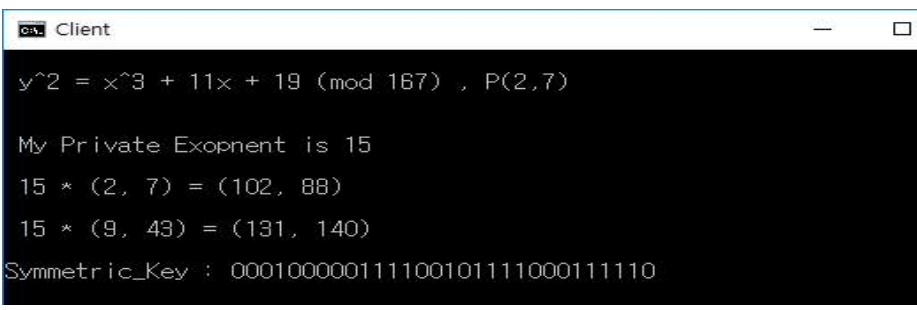
<p>클라이언트 ECC 디피-헬먼 키 교환 시연</p>	 <pre> <math>y^2 = x^3 + 11x + 19 \pmod{167}</math> , P(2,7)  My Private Exponent is 15 15 * (2, 7) = (102, 88) 15 * (9, 43) = (131, 140) Symmetric_Key : 000100000111100101111000111110 </pre>
<p>클라이언트 ECC 디피-헬먼 키 교환 시연 코드</p>	<pre> sock = Make_Client(RELAY_PORT, "127.0.0.1");  int Alice_Exponent = 15; int Symmetric_Key[BUFSIZE];  Key_Gender_ECC(sock, Symmetric_Key, Alice_Exponent);  printf("Symmetric_Key : "); for (int i = 0; i &lt; BUFSIZE; i++)     printf("%d", Symmetric_Key[i]); puts(""); </pre>

그림 4.4.3 클라이언트 측 ECC 디피-헬먼 키 교환 시연 화면

## 4.5 SHA-256

SHA-256 라이브러리는 KISA에서 배포하는 라이브러리를 활용하였으며, 메시지를 넣으면 해시된 메시지를 얻을 수 있는 외부에서 보기에는 간단한 구조이므로 따로 다이어그램으로 소개하지는 않는다. 대신 이번 절에서는 해시 함수의 동작 시연과 더불어 서버 측의 인증 정보 파일에서도 특히 ID와 PW를 해시하여 보관한 모습을 보인다. (해시와 더불어 다른 데이터를 위해 암호화도 병행하는 게 옳지만 우선 여기서는 해시함수의 시연이 주된 주제이므로 암호화는 따로 하지 않는다. 암호화는 앞서 소개한 모듈들을 사용하여 클라이언트, 릴레이 서버, 서버 측의 통신 할 메시지를 전송 시 암호화, 수신 시 복호화 해주면 쉽게 할 수 있기 때문에 우선은 생략한다.)


SHA-256 해시 시연	 <pre> Client Plain Text : HongGiIDong Hashed Text : eL헐j'&gt; ?.↔출퀵p철?}h_N}9뽕  Plain Text : Password Hashed Text : 靈&gt;涕 9샷0,oa.똥?+&amp;뉘N 9???→ </pre>
SHA-256 해시 시연 코드	<pre> strcpy(msg, "HongGiIDong"); printf("Plain Text : %s\n", msg); SHA256_Encrypt((BYTE*)msg, strlen(msg), (BYTE*)log); printf("Hashed Text : %s\n\n", log);  memset(log, 0, sizeof(log)); sprintf(msg, "Password"); printf("Plain Text : %s\n", msg); SHA256_Encrypt((BYTE*)msg, strlen(msg), (BYTE*)log); printf("Hashed Text : %s\n\n", log); </pre>

그림 4.5.1 SHA-256 시연 화면

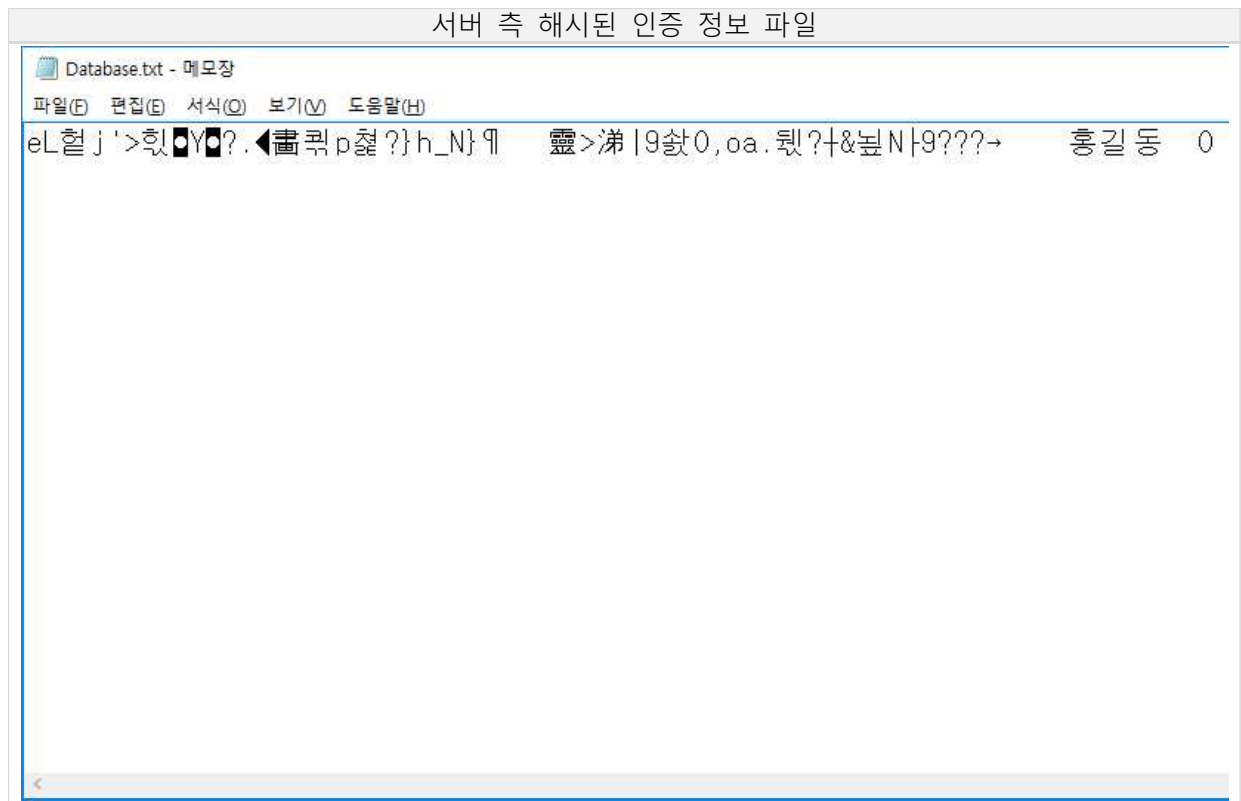


그림 4.5.2 SHA-256 시연 화면(Database.txt)

## 5 MiM(Man in the Middle) Attack Test

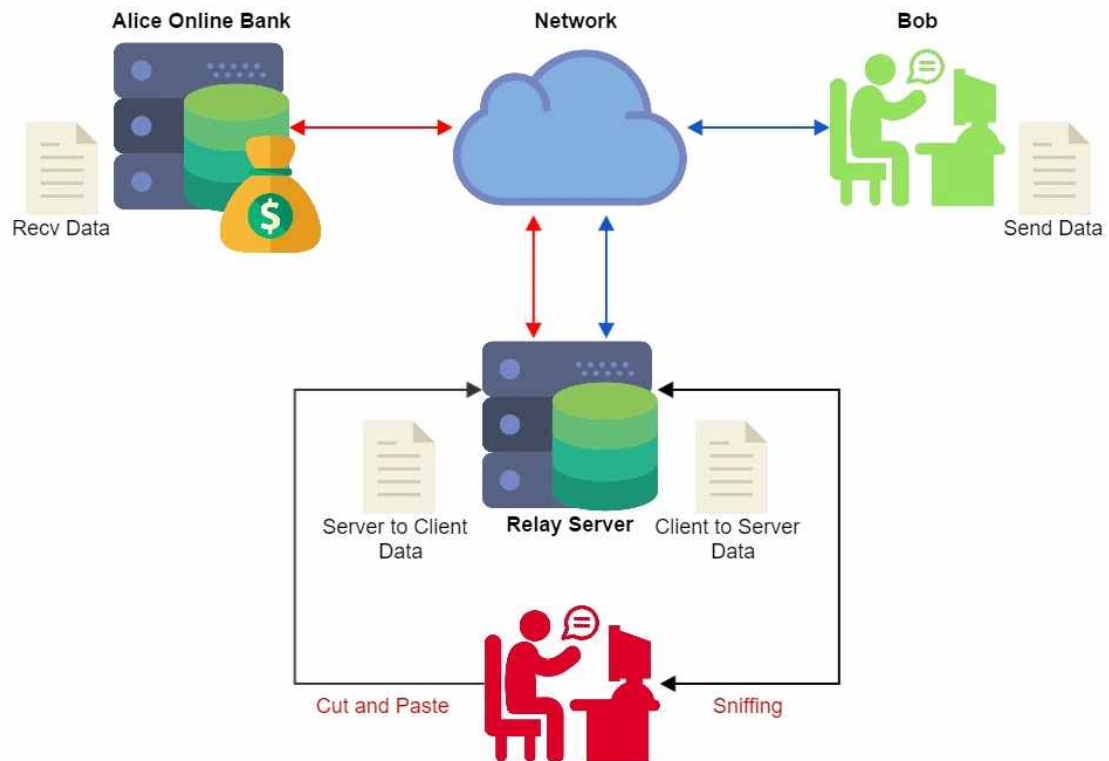


그림 5.1 Man in the Middle Attack Diagram

이번 장에서는 트루디가 릴레이 서버에 접속하여 클라이언트에서 서버로 전송되는 데이터를 변조해보는 테스트를 시연해본다. 테스트는 A5/1 스트림 암호화를 통해 암호화 전송 및 수신에 적용되고 ECC 디피-헬먼 알고리즘을 사용하여 키를 교환하는 시스템에서 시연된다. 그리고 서버 측 사용자 인증 정보 파일은 해시되지 않은 상태로 진행한다. 그 이유는 어떤 정보가 현재 들어있는지 보면서 시연하는 것이 이번 테스트 환경으로 적합하다고 판단했기 때문이다.

트루디의 악의를 품은 행위를 조금 더 간단하게 살펴보면 그림 5.2와 같다.

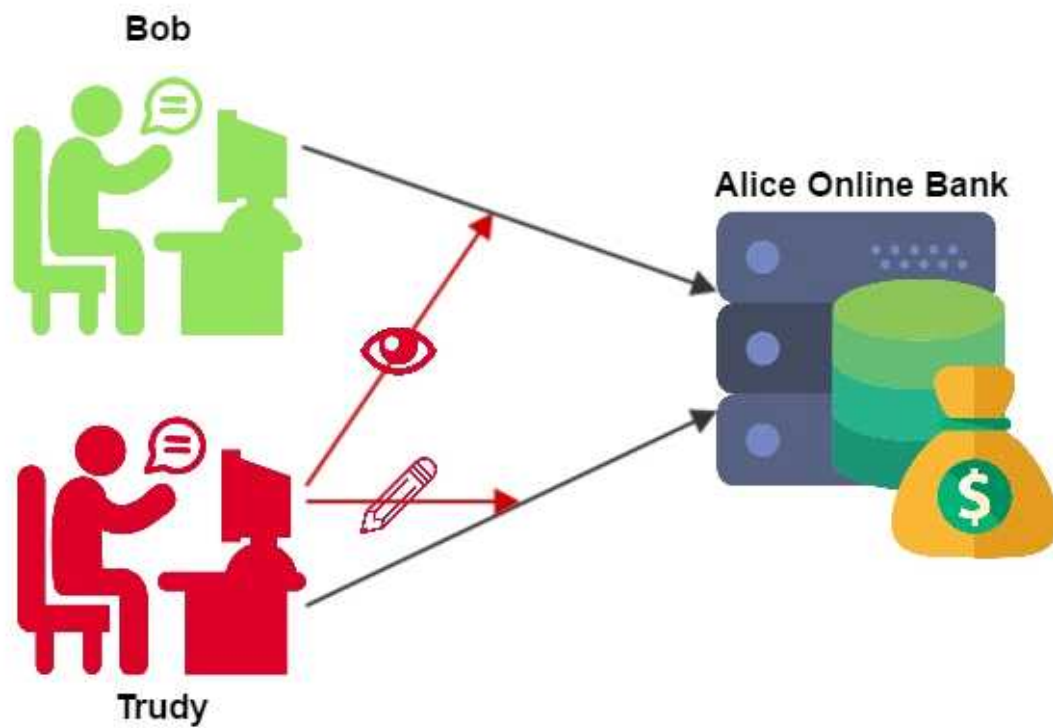


그림 5.2 Man in the Middle Attack Simple Diagram

Bob이 로그인 할 때 전송되는 정보를 트루디가 훔쳐보아 그 정보를 복사한 다음 자신이 로그인할 때 복사해 둔 정보를 Cut and Paste함으로써 자신이 Bob의 계정으로 로그인하는 공격법이다.



우선 그림 5.3은 Bob이 로그인할 때 트루디는 릴레이 서버에 접속하여 전송되는 정보를 훔쳐보는 화면이다.

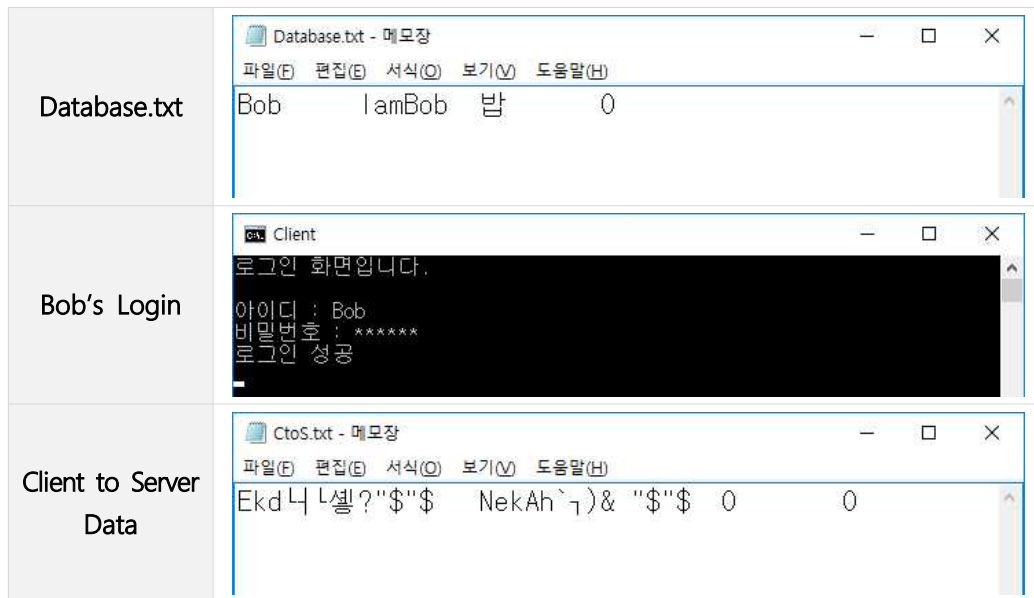


그림 5.3 Bob의 로그인 화면(Trudy가 훔쳐봄)

Bob's Login

Client to Server

TCP	552	49847→9000	[PSH, ACK]	Seq=1	Ack=1	Win=252	Len=512		
0020	50	18 00 fc e5 ba 00 00	45 6b 64 03 17 03 99 8f	P.....	Ekd.....				
0030	f0	7e 03 05 03 05 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21
TCP	552	49847→9000	[PSH, ACK]	Seq=513	Ack=1	Win=252	Len=512		
0020	50	18 00 fc 89 2b 00 00	4e 65 6b 41 68 60 02 08	P....+..	NekAh`..				
0030	07	01 03 05 7d 04 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21

Server to Client

TCP	552	49848→9001	[PSH, ACK]	Seq=1	Ack=1	Win=252	Len=512		
0020	50	18 00 fc 7d 0d 00 00	45 6b 64 03 17 03 99 8f	P...}...	Ekd.....				
0030	f0	7e 03 05 03 05 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21
TCP	552	49848→9001	[PSH, ACK]	Seq=513	Ack=1	Win=252	Len=512		
0020	50	18 00 fc 20 7e 00 00	4e 65 6b 41 68 60 02 08	P... ~..	NekAh`..				
0030	07	01 03 05 7d 04 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21	21 21 21 21 21 21 21 21

그림 5.4 WireShark를 통한 Bob의 로그인 패킷 분석

그림 5.4를 보면 클라이언트에서 릴레이 서버, 릴레이 서버에서 서버로 암호화된 ID와 PW가 전송되는 것을 볼 수 있다. 트루디는 이러한 정보를 릴레이 서버에 침입하여 가로챈다. 그 후 자신이 로그인할 때 전송되는 정보에 Bob의 암호화된 아이디와 패스워드를 붙여넣기 하여 전송한다. (실제 구현은 Sleep() 함수를 사용해 시간 딜레이를 준 다음 파일의 내용을 바꾸었다.)



Client to Server Data	<p>CtoS.txt - 메모장</p> <p>파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)</p> <pre>Svsg~1셀?N N  NekWuwfq•rL N  0 0</pre>
Trudy's Login	<p>Client</p> <pre>로그인 화면입니다.  아이디 : Trudy 비밀번호 : ***** 로그인 성공</pre>
Modified Client to Server Data	<p>CtoS.txt - 메모장</p> <p>파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)</p> <pre>EkdN 셀?"\$"\$ NekAh`~)&amp; "\$"\$ 0 0</pre>

그림 5.5 트루디의 Cut and Paste Attack 화면

Trudy's Login

Client to Server

```

TCP          552 50049->9000 [PSH, ACK] Seq=1 Ack=1 Win=252 Len=512
0020  50 18 00 fc 90 b0 00 00  53 76 73 67 7e 02 99 8f  P..... Svsg~...
0030  f0 7e 03 05 03 05 21 21  21 21 21 21 21 21 21 21  .~....!! !!!!!!!

TCP          552 50049->9000 [PSH, ACK] Seq=513 Ack=1 Win=252 Len=512
0020  50 18 00 fc c2 f9 00 00  4e 65 6b 57 75 77 66 71  P..... NekWufq
0030  07 01 03 05 01 04 21 21  21 21 21 21 21 21 21 21  .....!! !!!!!!!

```

Server to Client

```

TCP          552 50050->9001 [PSH, ACK] Seq=1 Ack=1 Win=252 Len=512
0020  50 18 00 fc 7d 0d 00 00  45 6b 64 03 17 03 99 8f  P...}... Ekd.....
0030  f0 7e 03 05 03 05 21 21  21 21 21 21 21 21 21 21  .~....!! !!!!!!!

TCP          552 50050->9001 [PSH, ACK] Seq=513 Ack=1 Win=252 Len=512
0020  50 18 00 fc 20 7e 00 00  4e 65 6b 41 68 60 02 08  P... ~.. NekAh`..
0030  07 01 03 05 7d 04 21 21  21 21 21 21 21 21 21 21  ....}.!! !!!!!!!

```

그림 5.6 Wireshark를 통한 트루디의 Cut and Paste Attack 패킷 분석

그림 5.6을 보면 클라이언트에서 릴레이 서버로의 메시지와 릴레이 서버에서 서버로의 메시지가 다른 것을 볼 수 있다. 이는 트루디가 릴레이 서버에 침입하여 데이터를 변조했기 때문이다. 따라서 트루디는 로그인 화면에서 서버에 등록되지 않은 정보를 입력했지만, 릴레이 서버에 접속하여 전송되는 데이터를 변조함으로써 Bob의 계정으로 로그인할 수 있게 되었다.

## 6 회고

이번 장에서는 현재까지 구현된 시스템의 보안상 문제점에 대해 알아보고, 구현하는 동안 어려웠던 점 그리고 배운 점에 대해 살펴본다.

현재(2016.10.30) 시스템은 암호화 전송 및 서버 측 사용자 인증 정보 파일의 해시 및 암호화까지 구현되었다.

우선 현재 시스템의 문제점은 '온라인 은행' 시스템의 서버는 사용자 클라이언트가 누구인지를 확인하는 인증 절차를 거치지만, 정작 중요한 클라이언트는 자신이 접속하는 서버가 정당한 '온라인 은행' 시스템의 서버인지 확인하는 인증 절차가 없다는 점이다.

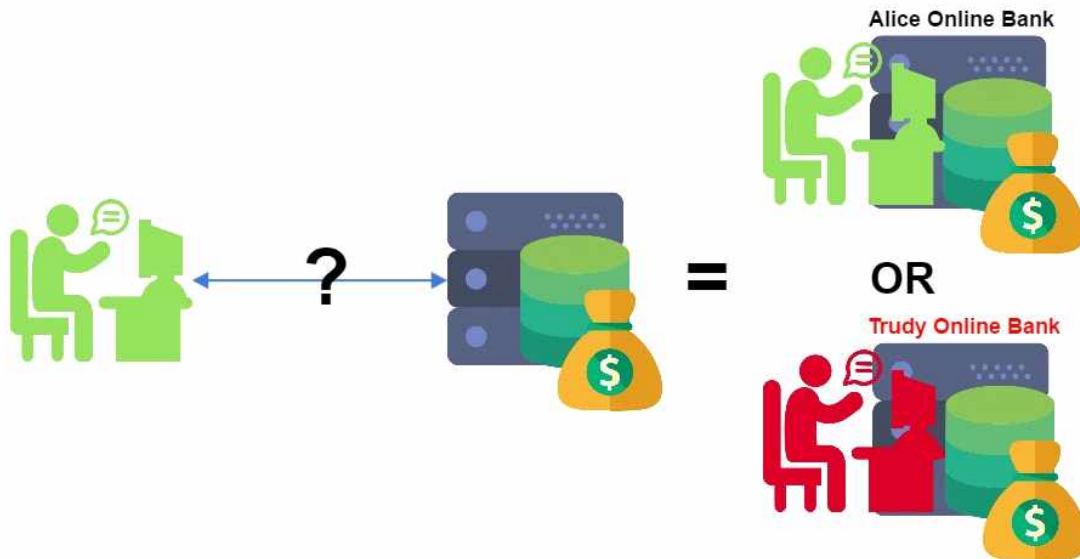


그림 6.1 서버 측 사용자 인증의 필요성

이 경우, 트루디가 실제 '온라인 은행' 시스템과 유사한 동작을 하는 서버를 만든다면 이는 보안상의 큰 문제가 될 것이다. 그러므로 서버 측의 인증 또한 필수적이다. 서버를 인증하는 절차도 시스템에 추가하여 이러한 '피싱' 문제를 해결해야 할 것이다.

그리고 인증된 서버라 할지라도 그들은 클라이언트의 패스워드를 알고 있고 개인의 패스워드는 통상 다른 여러 사이트들의 패스워드들과 연관성이 있다. 따라서 서버는 그 패스워드를 이용해 직접적 혹은 간접적으로 어떤 악의적인 행동을 취할 수 있다. 그리고 아무리 믿을만한 사이트라도 개인의 패스워드가 다른 곳에 알려지는 것은 결코 바람직하지 못한 상황이다. 이는 개인의 패스워드를 해시하여 서버에 전송함으로써 해결할 수 있다. 서버는 해시된 값만을 가지고 인증 절차를 진행하므로 개인의 패스워드는 그들조차 알 수 없다.

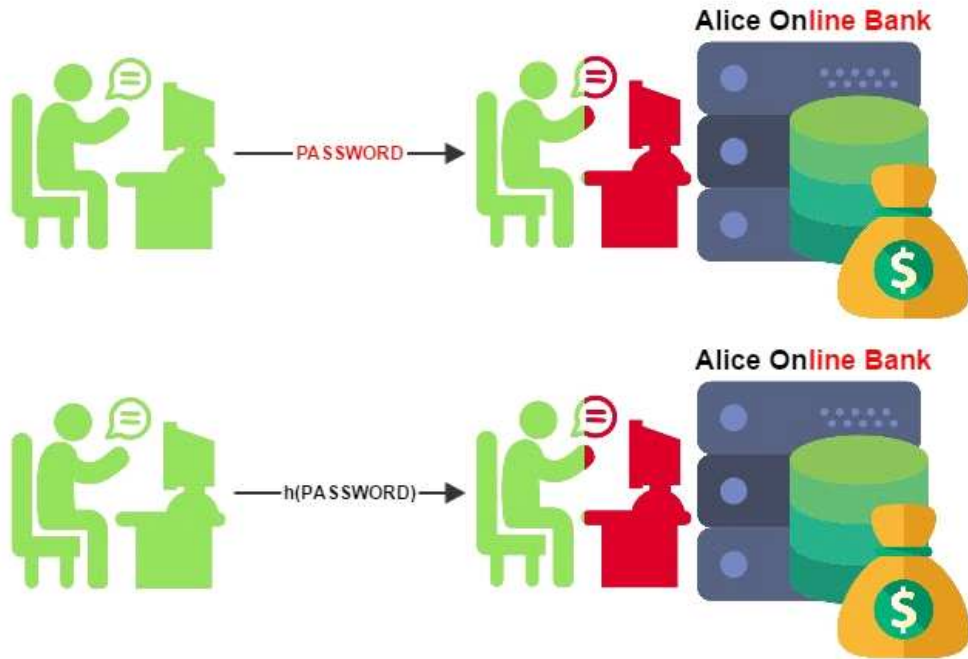


그림 6.2 신뢰할 수 없는 서버

하지만 5장 테스트에서 보았듯이 트루디가 중간에서 메시지를 변조하면 시스템의 규칙에 위반되는 상황을 여전히 쉽게 만들 수 있다. 5장 테스트에서는 전송 시 해시 함수가 적용되지는 않았지만, 이 중간자 공격, Cut and Paste 공격은 해시를 해서 보내도 결과는 같다. 대비책은 어떤 것이 있을까? 그 중 하나로 NONCE 기법이 있다. 서버에서 제공하는 NONCE라는 변칙적인 값을 개인이 패스워드를 해시할 때 첨가하여 해시하는 방법이다. SALT 방법과 유사하다. 이렇게 하면 5장 테스트와 같이 지나가는 정보를 복사해서 보관해놨자 다시 사용할 수 없게 되므로 Cut and Paste 공격을 막을 수 있게 된다.

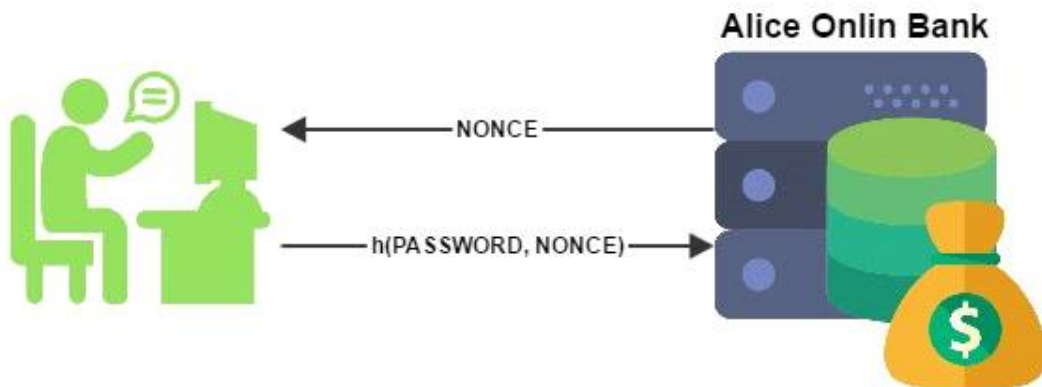


그림 6.3 NONCE 기법

하지만 이 방법은 서버 측이 개인의 패스워드를 알고 있는 상황에서 동작한다. 따라서 바로 이전에 말했던 문제점이 다시 제기되는 것이다. 그래서 이 문제와 트루디의 중간자 공격을 동시에 막는 것은 NONCE 값과 해시된 패스워드를 같이 해시함으로써 가능하다. 또한 이 부분은 추후 작성자가 추가할 보안 프로토콜이다.

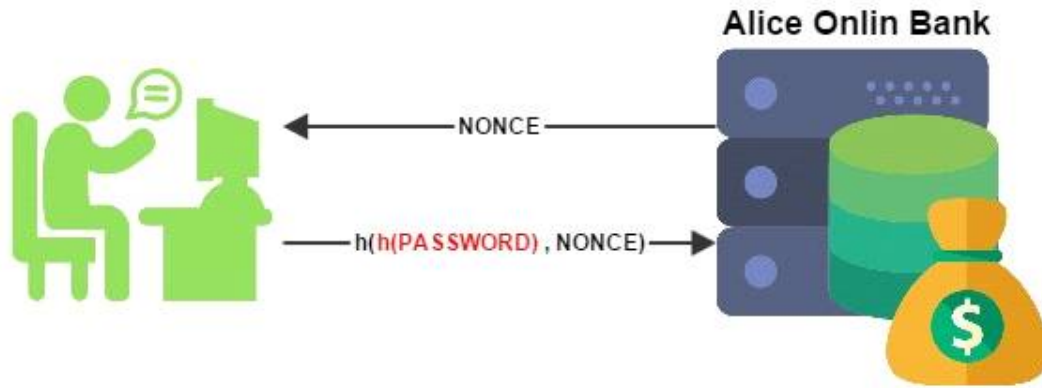


그림 6.4 Hashed NONCE 기법

그리고 아직 학습하지는 않았지만 소프트웨어의 취약점도 분명 존재한다. 버퍼 오버플로우 공격, 역어셈블 등이 가능할 것 같다. 또한 서버의 자원을 고갈시켜 서비스가 불가능하도록 하는 DoS 공격이 있을 수 있다. 이는 대비책을 세우기 굉장히 어려운데, L7 스위치가 DoS 공격을 어느 정도 막아낼 수 있다고 한다. 이 취약점들에 대한 조금 더 구체적인 대비책은 강의가 진행됨에 따라 학습할 것으로 보여 진다.

구현하면서 어려웠던 점은 일단 암호화 알고리즘을 적용할 때 비트 연산을 어떻게 프로그래밍에 적용 시킬지가 개인적으로 조금 문제가 되었다. 하지만 조금 비효율적이지만 직관적으로 이해하기 쉽게 주로 정수형 배열을 사용해 구현하였다. 또 실제 다른 컴퓨터로 프로그램의 동작을 테스트할 때 TCP의 경계 때문에 제대로 동작되지 않는 문제도 발생하였지만, 고정 길이로 데이터를 송신하고 수신함으로써 해결하였다. 또 서버 측의 해시된 정보 파일을 읽을 때 텍스트로 읽어서 문제가 생겼지만 이는 바이너리로 읽음으로써 해결하였다. 또 비슷한 문제로 문자열을 암호화하니 간간히 암호문 사이에 원할한 동작을 방해하는 특수 문자가 들어가는 경우가 생기는 것을 운 좋게 파악하여 조건문으로 해결할 수 있었다.

이번 과제를 진행하면서 여러 암호화를 직접 구현해볼 수 있었으며, 직접 트루디가 되어 중간에서 정보를 변조하는 테스트도 해볼 수 있었다. 조금씩 진행되는 프로젝트 덕분에 전체적인 보안 개념을 학습하는 데 큰 도움이 되고 있다. 앞으로의 여러 인증 프로토콜 및 현재 사용되어지는 프로토콜들, 그리고 소프트웨어의 취약점에 대한 학습 및 프로젝트의 향후 진행 방향이 정말 기대되는 부분이다.

## 7 부록

이번 장에서는 카이사르 암호에 대해 소개하고 암호문의 빈도수를 검사하여 Breaking하는 프로그램을 보인다. 우선 카이사르 암호는 고전 암호 중 하나로서 1~25까지 25개의 키를 가진다. 암호화는 평문의 알파벳에 대하여 키의 값만큼 시프트 함으로써 수행되고, 복호화는 반대 방향으로 시프트 함으로써 수행된다. 이 암호화는 키의 범위가 매우 한정적이라는 점에서 취약하다. 따라서 전수검사의 비용이 크지 않지만 영 문장의 특성상 알파벳의 빈도수가 노출된다는 점을 흥미롭게 생각해서 프로그램으로 작성하여보았다.

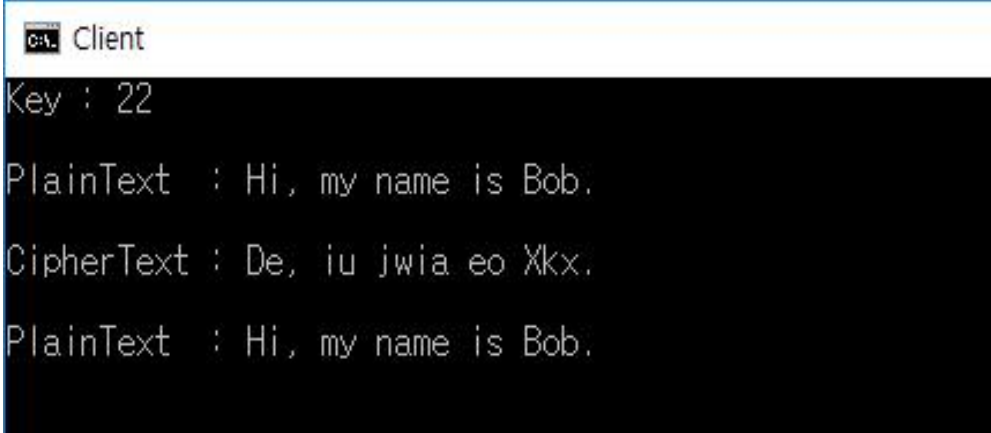
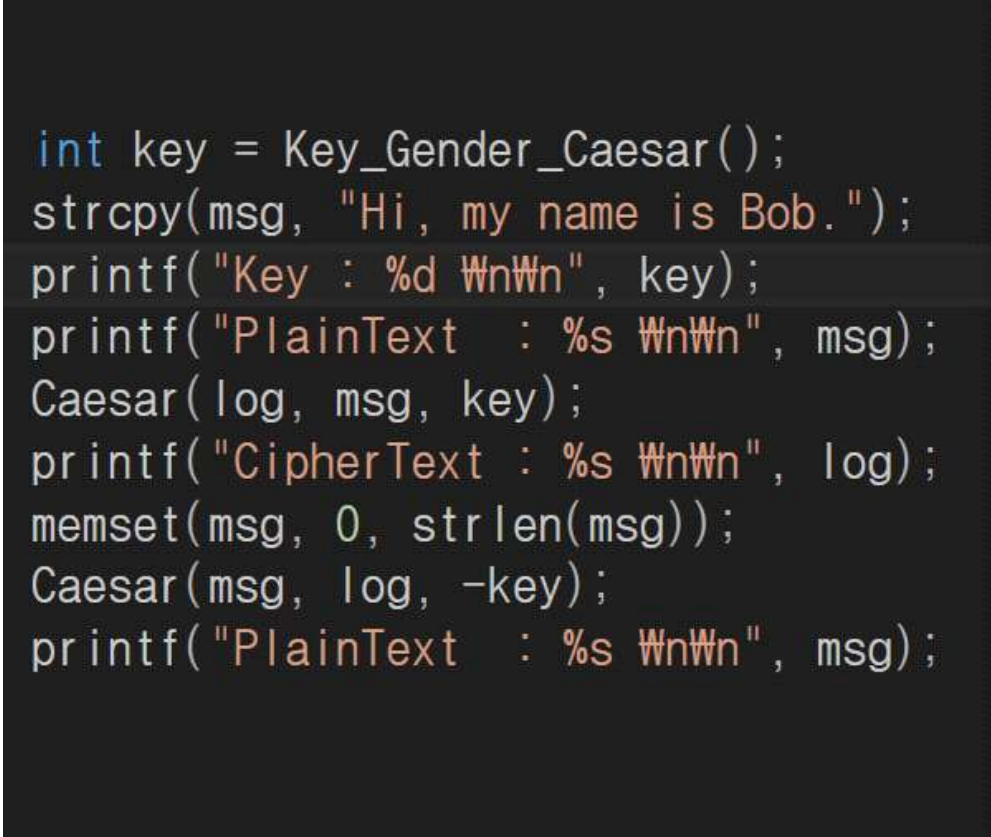
카이사르 암호화 시연	 <pre>Client Key : 22 PlainText : Hi, my name is Bob. CipherText : De, iu jwia eo Xkx. PlainText : Hi, my name is Bob.</pre>
카이사르 암호화 시연 코드	 <pre>int key = Key_Gender_Caesar(); strcpy(msg, "Hi, my name is Bob."); printf("Key : %d \n\n", key); printf("PlainText : %s \n\n", msg); Caesar(log, msg, key); printf("CipherText : %s \n\n", log); memset(msg, 0, strlen(msg)); Caesar(msg, log, -key); printf("PlainText : %s \n\n", msg);</pre>

그림 7.1 카이사르 암호 시연 화면

Key : 22

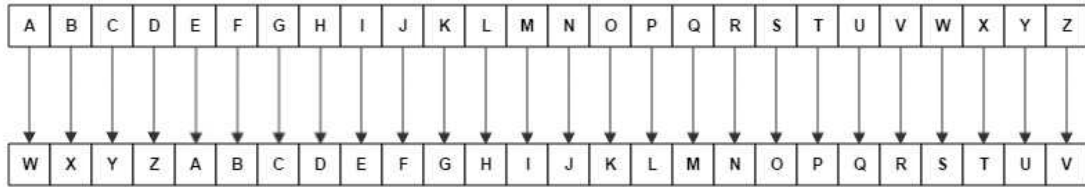


그림 7.2 key가 22일 때 카이사르 암호

이번에는 빈도수를 이용한 카이사르 암호 Breaking을 해보자. 우선 키는 난수로 지정해야 하지만 이번 테스트에서는 위의 다이어그램을 활용하기 위해 키 값을 22로 하여 에드거 앨런 포의 The Gold Bug 소설을 암호화한다.



그림 7.3 The Gold Bug 소설을 카이사르 암호화한 결과



이제 암호화가 된 텍스트 파일의 알파벳 빈도수를 프로그램을 사용해 세어볼 것이다. 그 전에 우리는 다음에 보이는 자료에서처럼 영어의 문장에서 알파벳 e가 가장 빈도수가 높은 것을 알고 있다. 참고로 아래 자료는 1982년 영국의 Beker가 조사하여 발표한 자료이다.

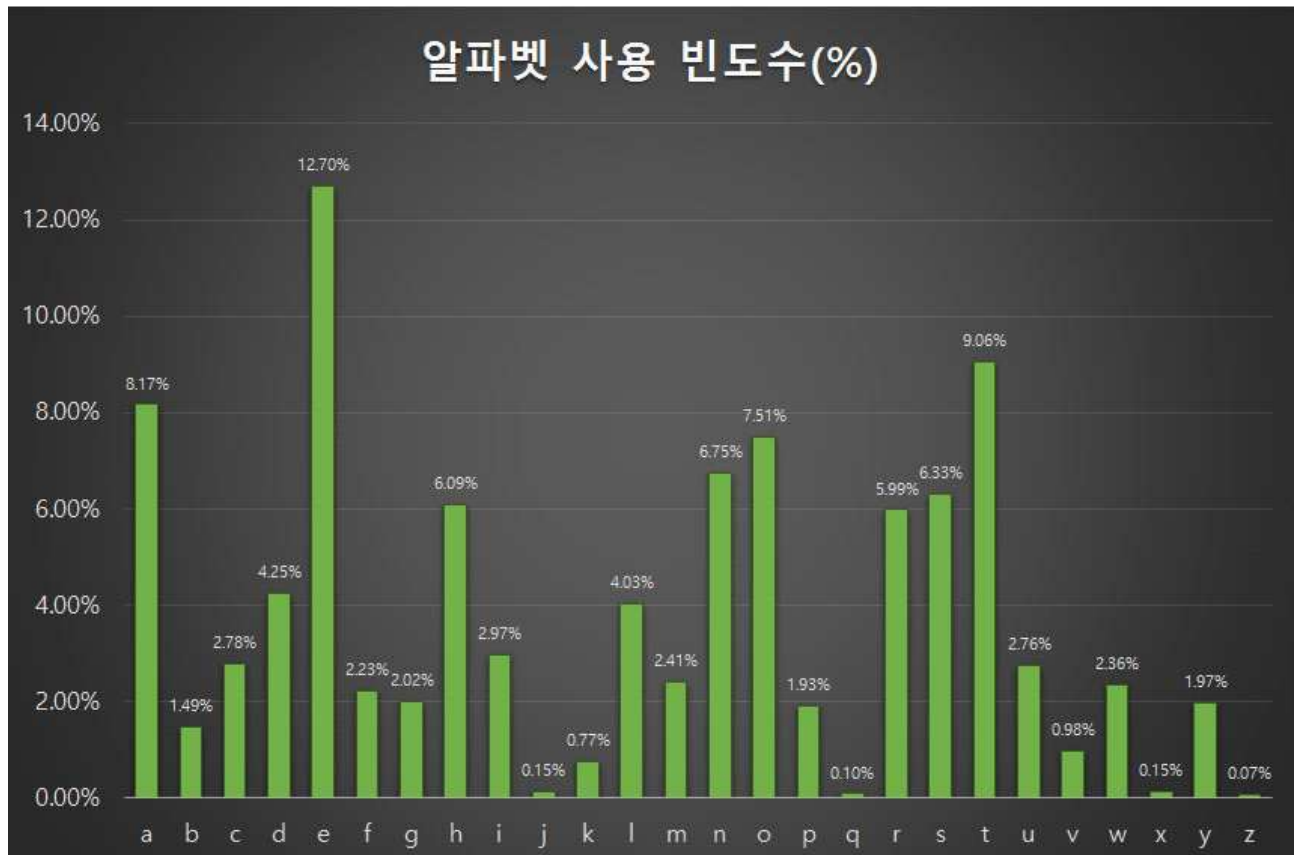


그림 7.4 알파벳 사용 빈도수(%)

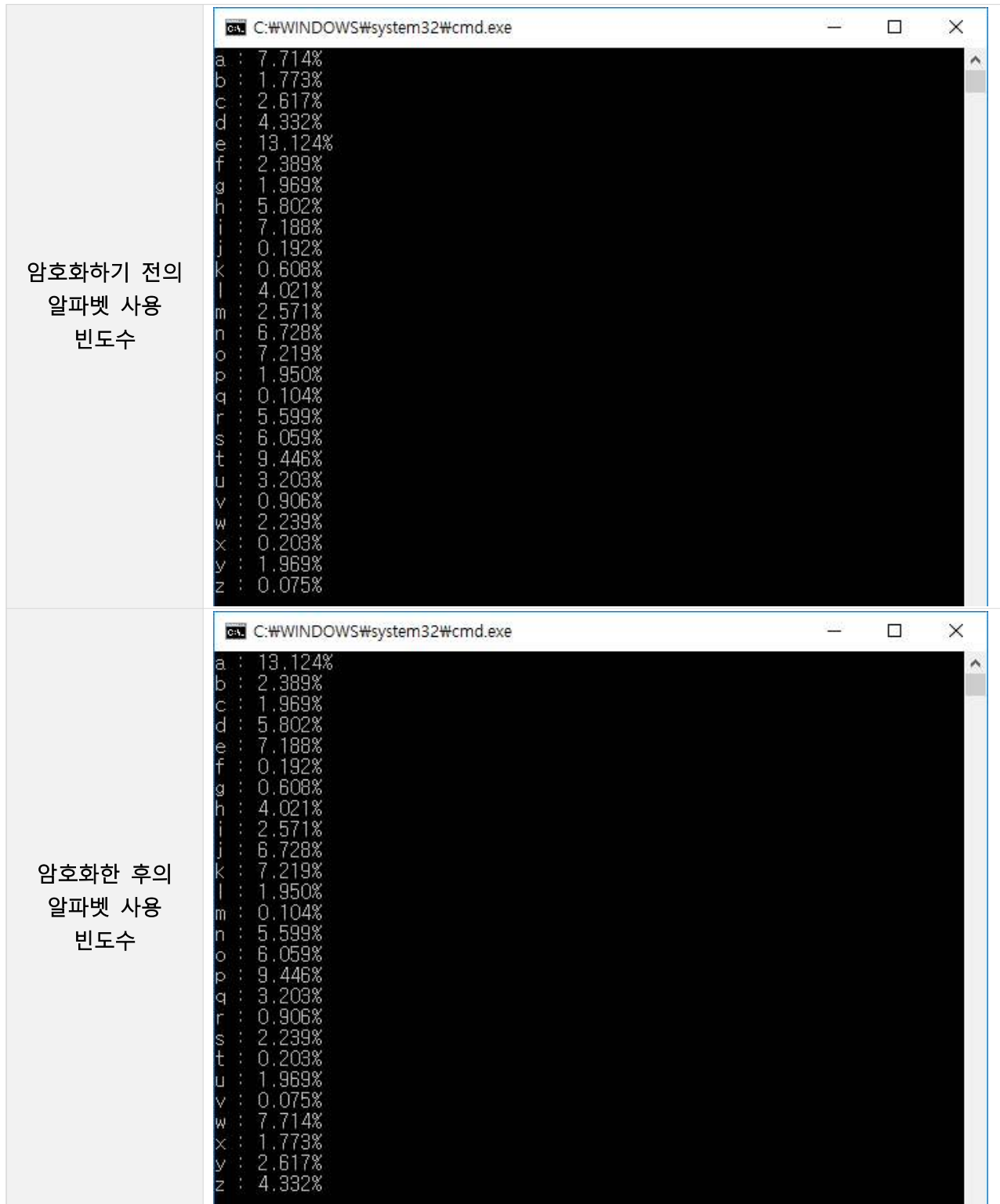


그림 7.3 The Gold Bug 소설 알파벳 사용 빈도수

위의 결과를 보면 암호화 후에 a의 빈도가 가장 많은 것을 볼 수 있다. 따라서 우리는 e가 a로 치환되었다는 것을 알 수 있고, 곧바로 키의 값이 22라는 것을 유추할 수 있을 것이다.