정보보호 기술을 적용한 Application 개발

1:N 채팅 프로그램

정보보호

12조

201420924 정민규
201520882 천윤서
201620927 엄지혜
201720728 현광빈
201720797 최석영

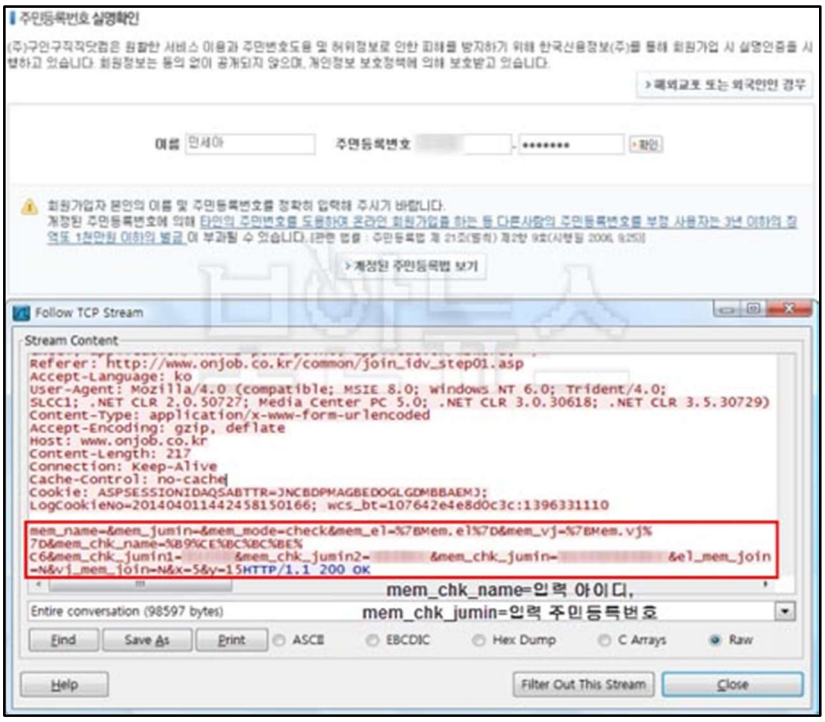
목차

:

1. 개요

1) 문제점

파일 공유 사이트를 비롯한 많은 웹사이트에서 로그인 시 중요 정보가 암호화되지 않은 상태로 전송되는 문제점이 제기되고 있다. 해당 취약점이 패치되지 않은 상황에서 같은 네트워크를 사용하는 다른 사용자가 로그인 과정을 스니핑할 경우, 로그인하는 사용자 계정을 탈취해 2차 피해가 발생할 수 있다. 따라서 로그인 정보를 암호화 하여 서버로 전송할 수 있도록 신속한 조치가 필요하다.[1] 또한 로그인 정보 외에도 메시지 내용까지 다른 사용자가 볼 수 있게 된다.

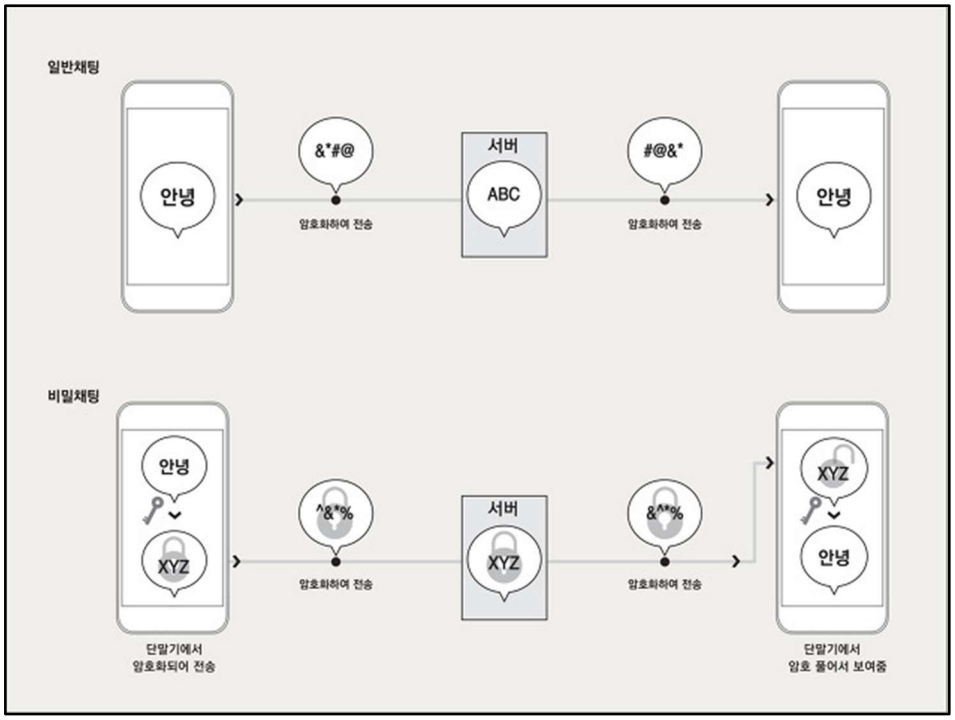


[그림1-1-1] 구인구직닷컴(http://www.onjob.co.kr/)에서 실명확인 시 주민등록번호 평문 전송

작년 드루킹 사건 이후로 메신저의 보안성이 이슈화되고 있다. 사람들은 이제 메신저의 각 기능 뿐만 아니라 보안성을 가장 주요한 요소로 뽑는다. 2014년 카카오톡에서도 종단간 암호화를 적용한 비밀 채팅방 기능을 적용했다. 비밀 채팅방에서 나눈 대화가 암호화 되고, 그 암호화를 풀 수 있는 키가 대화에 사용자 스마트폰 안에 저장되어있어 대화 당사자 외에는 아무도 들여다볼 수 없게 되었다.

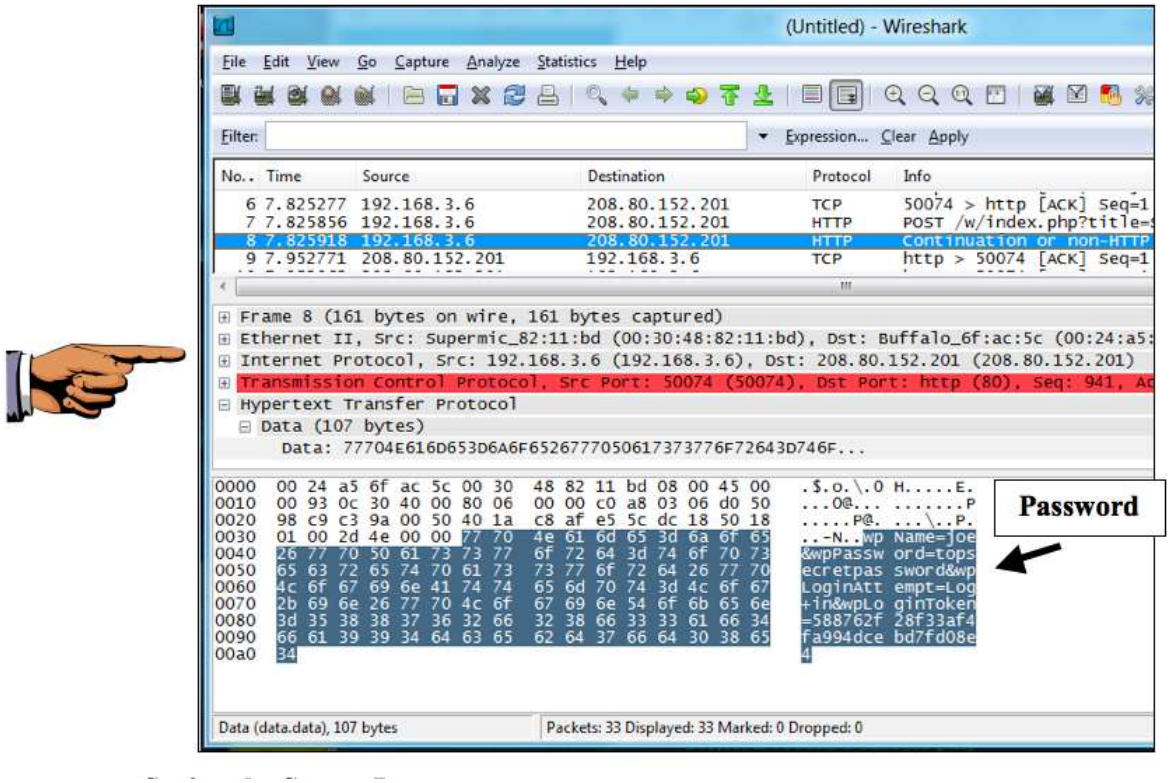
---

[1] https://www.boannews.com/media/view.asp?idx=40472



[그림 1-1-2] 카카오톡 비밀 채팅 기능

2) 목적



[그림 1-2-1] wireshark을 통해 패스워드를 도청한 모습

채팅 프로그램은 보안 기능을 적용하지 않는다면 WireShark등의 유틸리티로 쉽게 중간에 누군가가 메세지를 훔쳐 볼 수 있을 것이다. 또한, 로그인을 할 때 비밀번호를 그대로 전송한다면, 악의적인 사용자가 평문으로 노출된 비밀번호와 사용자 계정을 알고, 채팅방에서 사칭을 할 수도 있을 것이다.

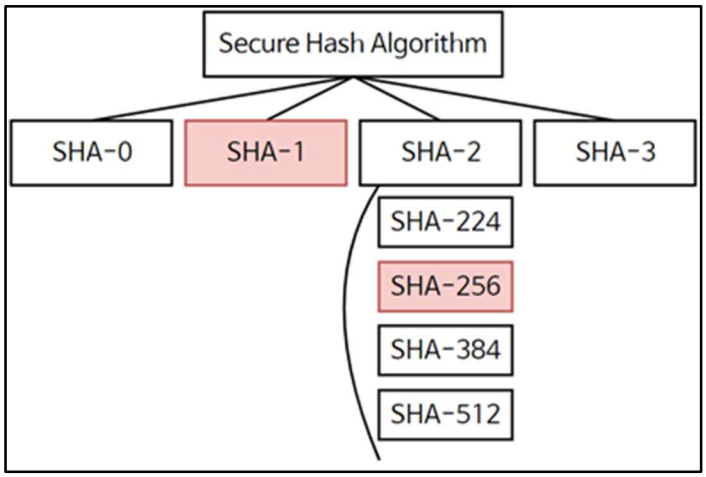
이번 프로젝트의 목적은 다수의 사용자가 사용할 수 있는 로그인 기능을 가진 채팅 프로그램을 보안 기능을 적용하여 만드는 것이다. 즉, 다수의 사용자가 하나의 서버에 붙어서 도청의 위험 없이 채팅할 수 있는 환경을 만드는 것이 목적이다.

채팅 프로그램에 접속할 사람들은 암호화된 인증 과정을 거칠 것이고, 채팅 방에 있는 사람들은 암호화된 메세지를 1:N으로 서로 주고받아서 채팅을 할 수 있는 것이 궁극적인 목표이다.

더불어, 시연을 하기 위해서 평문을 그냥 보내는 기능과, 암호화된 메세지를 보내는 기능을 따로 분리할 계획이다.

2. 암호화 알고리즘

1) SHA 256



[그림 2-1-1] SHA 알고리즘 분류

SHA(Secure Hash Algorithm)는 서로 관련된 암호학적 해시 함수들의 모음이다. 이들 함수는 미국 국가안보국(NSA)이 1993년에 처음으로 설계했으며 미국 국가 표준으로 지정되었다. SHA의 최초의 알고리즘은 1993년에 미국 표준 기술 연구소(NIST)에 의해 안전한 해시 표준으로 출판되었으며, 다른 함수들과 구별하려 보통 SHA-0으로 부른다.

그 후 NSA는 1995년에 개정된 알고리즘(FIPS PUB 180-1)을 새로 출간 했으며 이를 SHA-1로 부른다. 현재 SHA-1은 SHA 함수들 중 가장 많이 쓰이며, TLS, SSL, PGP, SSH, IPSec 등 많은 보안 프로토콜과 프로그램에서 사용되고 있다. SHA-1은 이전에 널리 사용되던 MD5를 대신해서 쓰이기도 한다. SHA-1은 SHA-0의 압축 함수에 비트 회전 연산을 하나 추가한 것이다. SHA-0과 SHA-1은 최대 264비트의 메시지로부터 160비트의 해시값을 만들어 내며, 로널드 라이베스트가 MD4 및 MD5 해시 함수에서 사용했던 것과 비슷한 방법에 기초한다.[2] 하지만 SHA-1은 최근 2017년 구글(Google)이 주도적으로 진행한 해시 보안 관련 연구에서 충돌이 감지되어 신규 프로젝트에는 점차 차용 되지 않는 추세다.[3]

NIST는 나중에 해시값의 길이가 더 긴 네 개의 변형을 발표했으며, 이들을 통칭하여 SHA-2라 부른다. SHA-256, SHA-384, SHA-512는 2001년에 초안으로 처음으로 발표되었으며, 2002년에 SHA-1과 함께 정식 표준(FIPS PUB 180-2)으로 지정되었다. 2004년 2월에 삼중 DES의 키 길이에 맞춰 해시값 길이를 조정한 SHA-224가 표준에 추가되었다. 결과적으로

---

[2] https://ko.wikipedia.org/wiki/SHA

[3] http://www.bitweb.co.kr/news/view.php?idx=388

현재 가장 많이 사용되고 있는 해시 알고리즘은 SHA-256이며 상당수 블록체인에서 채택하여 사용하고 있다.

SHA-256은 256비트로 구성되어 어떤 문자열이 들어오던 64자리 문자열을 반환한다. MD5는 32자리 문자열을 반환한다는 점과 비교했을 때 해시 값이 2배나 길다는 특징을 가진다. 단순히 산술적으로 계산했을 때 256비트는 2의 256 제곱만큼 경우의 수를 만들 수 있다. 이는 개인용 컴퓨터로 무차별 대입(Brute-Forcing)을 수행해 해시 충돌 사례를 찾으려고 할 때 억겁의 시간이 소요될 정도로 큰 숫자이므로 충돌로부터 비교적 안전하다고 볼 수 있다.

예를 들어 "안녕하세요"를 SHA-256로 해시 시키면
2C68318E352971113645CBC72861E1EC23F48D5BAA5F9B405FED9DDDCA893EB4라는 결과를 얻을 수 있고 "안녕"을 SHA-256으로 해시 시키면
BB5E5015D86D85847ACC1742E4A339B40872F50C51BC4215341778CE9F945CD5
라는 결과를 얻을 수 있다.

로그인 과정에서 패스워드를 그대로 노출하면, 공격자가 패스워드를 가져갈 수 있는 위험성이 존재한다. 이번 프로젝트에서는 이러한 패스워드 노출을 막기 위해서 SHA 알고리즘을 사용하기로 하였다.

2) 대칭키 알고리즘

암호화 알고리즘은 크게 두 종류가 있다. 하나는 대칭키 알고리즘이고, 다른 하나는 비대칭키 알고리즘이다. 그 중에서 대칭키 알고리즘은 암호화를 하는 쪽과 복호화를 하는 쪽이 서로 같은 키를 쓰는 알고리즘이다. 대칭키 알고리즘은 비대칭키 알고리즘과 다르게 수학적 이론에 의존하는 것이 아닌, 단순한 복잡성에 의존한다. 따라서 대칭키 암호화 알고리즘은 비대칭키 알고리즘에 비해 간단하며, 속도가 훨씬 빠르다.

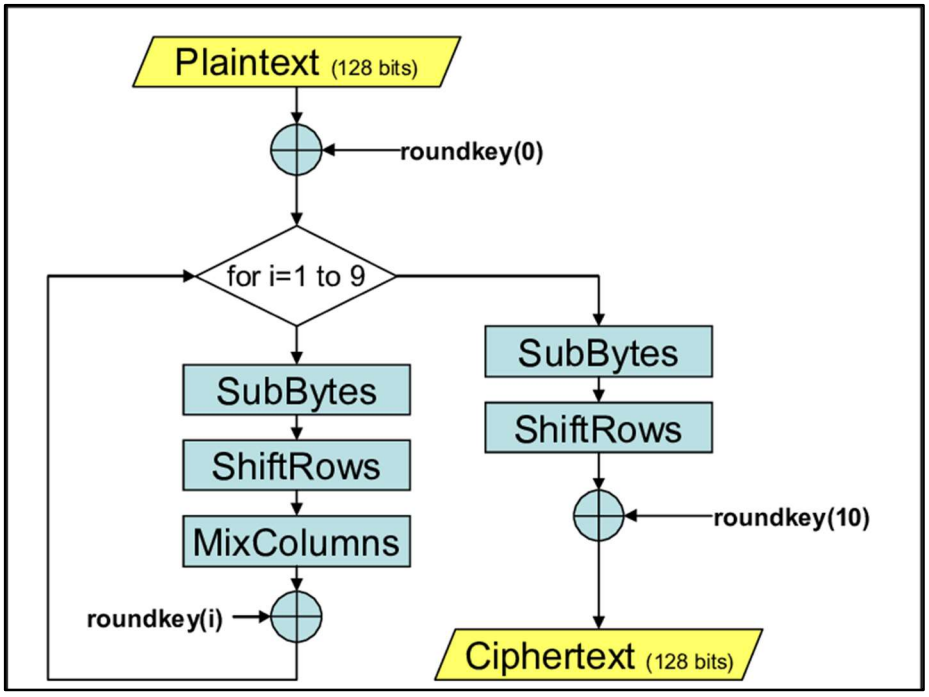
	DES	3DES	AES
Key length(bits)	56	112 or 168	128, 192, or 256
Key strength	Weak	Strong	Strong
Processing requirements	Moderate	High	Low
Ram requirements	Moderate	Moderate	Low
Remarks	Created in the 1970s	Applies DES three times with 2 or 3 keys	Today's gold standard, likely to be dominant in the future

<표: DES, 3DES, AES 의 비교>[4]

일반적으로 대칭키 암호화 알고리즘에는 AES, DES, 3DES가 있다. 먼저 DES는 과거에 쓰이던 방식으로 1970년도에 개발되었다. 하지만, 키 길이가 56 bits로 매우 짧기에 현재는 보안에 문제가 있어 잘 쓰이지 않는다. 이러한, 키 길이의 문제를 극복하기 위한 중간 단계로서 3DES가 등장했다. 3DES는 DES 알고리즘을 3번 적용시키는 식으로 동작한다. 현재 가장 많이 쓰이는 표준에 가까운 대칭키 암호화 알고리즘은 AES이다. AES는 DES의 뒤를 이을 표준을 채택하기 위해서 NIST가 주최한 공모전에서 채택한 레인달(Rijndael) 알고리즘이다. 128, 192, 256bit의 키 길이를 선택할 수 있고, 시스템의 리소스를 크게 소모하지 않는다. AES는 가장 널리 쓰이는 강력한 보안성을 지닌 표준이고, 리소스에 부담을 덜 주는 대칭키 암호화

[4] Randall J Boyle and Raymond R Panko, *Corporate Computer Security* (n.p.: PEARSON, 2015), 167.

알고리즘이기에, 이번 프로젝트의 메세지를 직접 암호화 하는 부분에서 AES 알고리즘을 채택하기로 하였다.



[그림 2-2-1] AES의 간략한 구조.

AES는 먼저 KeyExpansion 과정을 거친다. AES는 여러 단계의 라운드를 가지고 있는데, 하나의 암호화 키를 가지고, 해당 라운드 들에 사용할 여러 라운드 키를 생성한다. AES는 각 라운드에서 크게 4가지 작업을 한다. S-Box, Shift Row, Mix Column, Add Round Key.

먼저 AES에 들어간 암호화할 평문의 블럭은, 미리 주어진 S-BOX(Substitution BOX)를 통과한다. S-BOX가 하는일은 간단한데, 각 바이트를 보고 S-BOX의 행과 열이 일치하는 곳의 새로운 바이트로 치환을 하는 것이다.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

[그림 2-2-2] : S-Box

예를 들어, 위 그림에서 0x43은 40과 03이 만나는 열인 0x1a로 치환이 된다.

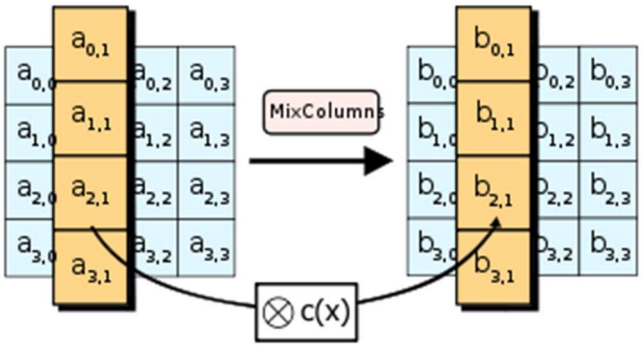
ShiftRows에서는 암호화할 블럭을 각 행 마다 왼쪽으로 민다. 예를 들어 아래와 같은 블럭이 있다고 하자.

0x03	0x02	0x1A	0x3B
0x4B	0xFF	0x23	0x11
0x63	0x88	0x4A	0x10
0x66	0x0E	0x74	0x99

해당 블럭의 두 번째 칸은 두 칸, 세 번째 칸은 세칸, 네 번째 칸은 네칸 씩 밀리면서 아래와 같이 바뀐다.

0x03	0x02	0x1A	0x3B
0xFF	0x23	0x11	0x4B
0x4A	0x10	0x63	0x88
0x99	0x66	0x0E	0x74

그 다음 MixColumn 단계에서는 열 단위로 섞기 위하여 각 열을 어떤 행렬 c(X)와 연산을 하고, 마지막으로 AddRoundKey에서 RoundKey와 암호화할 블럭을 XOR연산을 해서 ROUND를 마친다.



[그림 2-2-3] MixColumn

3) RSA

RSA(Rivest Shamir Adleman) 공개키 알고리즘은 현재 공개키 암호화에서 가장 널리 쓰이고 있는 공개키 알고리즘이다. RSA 공개키 알고리즘의 기본 형태는 DES와 같은 대칭키 암호화와 같다. 키를 사용하여 평문을 암호화시켜 암호문을 출력한다. 여기서 키란 메시지를 열고 잠그는 상수를 의미하는데 자신의 개인키로 암호화하여 공개키로 복호화 할 수 있다. 공개키 암호는 정보 송신자와 수신자가 서로 다른 키(공개키, 개인키)를 사용하므로 비대칭키 암호 방식이라고도 하며, 사전에 개인키를 나눠갖지 않아도 안전한 정보 송수신이 가능하다는 장점이 있다.

RSA는 큰 소수의 곱으로 이루어진 정수의 소인수분해가 매우 어렵다는 사실을 이용하여 안정성을 높인다. Mod 연산을 통한 큰 숫자의 암호화 연산은 비교적 쉽지만, 그 역을 추적해가는 것은 매우 어렵기 때문에 RSA 방식의 안전도 보장된다. RSA는 소인수분해의 난해함에 기반하여, 공개키만을 가지고는 개인키를 쉽게 짐작할 수 없도록 디자인되어 있다.

RSA는 이러한 공개키와 비밀키를 아래와 같은 과정을 통해 구한다.

1. 두개의 서로 다른 소수 (p, q)를 선택한다.
2. 두 수를 곱하여 $N = p\ x\ q$의 값을 구한다.
3. $\phi(N) = (p - 1)(q - 1)$을 구한다.
4. $1 < e < \phi(N)$이고 $\phi(N)$와 서로소인 정수 e를 선택한다.
5. 유클리드 호제법을 이용하여 $e\ x\ d\ = 1(mod\phi(N))$를 만족하는 d를 구한다.

위의 과정을 거쳐서 계산된 값 중, 공개키는 $< e, N >$이며, 개인키는 $< d >$가 된다. 만약 서로 다른 소수 (p, q)가 알려진다면, 이를 통해 생성된 공개키와 비밀키가 쉽게 노출될 위험성이 존재한다.

RSA는 보안 강도가 높고 키 분배, 키 관리가 굉장히 효율적이다. 그러나 대부분의 비 대칭형 암호방식이 그러하듯, 대칭형 암호 방식에 비해 시간이 많이 소요된다.

AES를 이용해서 암호화 하는 쪽과 복호화 하는 쪽은 둘 다 동일한 키를 가지고 있어야 하는데, 그를 위해서 AES키를 교환하는 과정에서 키가 도청 될 가능성이 존재한다. 이번 프로젝트에서 RSA는 이러한 대칭키 교환 과정에서 적합하다고 생각하여 채택하였다.

3. 프로그램 설명

1) DB

사용자 간 채팅을 구현하기에 앞서, 로그인 서비스를 제공하기 위해 User table을 생성하였다. User table에 등록되어있는 사용자의 id, password를 각각 데이터베이스에 저장하였고, user_salt를 통해 여러 유저가 같은 암호를 사용하더라도 각자 고유한 salt를 가지고 Hashing을 진행해 Attacker가 암호를 쉽게 알지 못하게 구현하였다.

index	user_id	user_pw	user_salt
1	kim	7CC9311CFB06A3B09C96F1FC1F32BDDCA2A8...	IVbtGWtB@w[8=3A_
2	park	542B048CBBF48D586C7AF3D307160A953BBCE...	Y@EM_BCaJHc]7^[Z
3	cho	E14778A55F77741FDB10275C0FEB9F8899B2A...	VnZRsDPIjW@WZBbW

[그림 3-1-1] 데이터베이스 내 user table

서버는 데이터베이스를 DBManager.java를 이용하여 접근한다. DBManager에서 데이터베이스의 접근 계정의 id, password를 이용하여 해당 데이터베이스 스키마에 접근한다. 본 프로젝트에서는 사용자간 채팅 서비스를 주 목적으로 하였기에 데이터베이스에 접근하는 부분은 별도의 암호화과정을 거치지 않았다.

```
public class DBManager {
    public static String DB_URL = "jdbc:mysql://localhost:3306/crypto?serverTimezone=UTC&useSSL=false"; // db 이름
    public static String DB_USER = "root"; // db userid, pwd 설정
    public static String DB_PASSWORD = "1234";

    public static int get_table_from_DB(String id, String password) {

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // 드라이버 로딩
            Class.forName("com.mysql.cj.jdbc.Driver");

            conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);// db연결설정

            stmt = conn.createStatement(); // Statement를 가져옴
            String query="SELECT user.index from user where user_id='"+id+"' and user_pw='"+password+"'";
            rs = stmt.executeQuery(query.toString()); // SQL문 실행

            while (rs.next()) { // query 출력 및 실행
                String result=rs.getString(1); //첫번째 값이 넘어옴
                System.out.println("result: "+result);
                return Integer.parseInt(result);
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                rs.close();
                stmt.close();
                conn.close();
            } catch (SQLException e) {

            }
        }
        return -1;
    }
```

[그림 3-1-2] DBManager.java

DBManager에서 클라이언트의 정보를 바탕으로 서버는 테이블을 검색해 해당 유저가 있는지 탐색한다. user가 있는 경우 해당 유저의 index를 반환하여 사용자가 데이터베이스에 저장되어 있는지를 판별한다.

2) 채팅

- 로그인 서비스

1 : N 채팅을 진행하기에 앞서, 사용자는 서버에게 로그인을 요청한다. 사용자의 정보는 앞서 설명한 Database에 user table에 존재한다. 로그인을 위한 user_id, user_password attribute가 사용자가 입력한 아이디, 패스워드와 일치한다면 정상적으로 채팅방에 진입한다.

Client의 sendMsg()는 UI에서 전해진 message를 상황별로 구분해 로그인과 채팅을 진행한다. 로그인을 진행하는 경우, 서버에게 LoginRequest라는 tag와 함께 입력된 user_id와 user_password를 전송한다. 서버는 응답을 받아 로그인 요청을 진행한다.

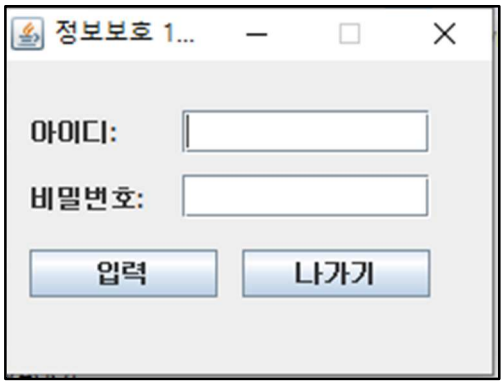
```
public void sendMsg() throws Exception {

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(socket.getOutputStream(), true);
        if (cf.isFirst == true) {
            System.out.println("First session Connected");
            InetAddress iaddr = socket.getLocalAddress();
            String ip = iaddr.getHostAddress();
            this.getUserStatus();
            System.out.println(user_password);
            pw.println("LoginRequest_" + user_id + "_" + user_password);
            System.out.println("ip:" + ip + "id:" + user_id);
            str = "[" + user_id + "] 님 로그인 (" + ip + ")";
        } else {
            String plainText = cf.txtF.getText();
            if (this.cipher_check == false) {
                str = "[" + user_id + "]: " + plainText;
                pw.println("Message_" + str);
            } else if (this.cipher_check == true) {
                String cryptText = aes.aesEncode(plainText);
                str = "[" + user_id + "]: " + cryptText;
                pw.println("Cipher_" + str);
            }
        }

    } catch (IOException ie) {
        System.out.println(ie.getMessage());
    } finally {
        try {
            if (br != null)
                br.close();
        } catch (IOException ie) {
            System.out.println(ie.getMessage());
        }
    }
}
```

[그림 3-2-1] Client의 sendMsg() method

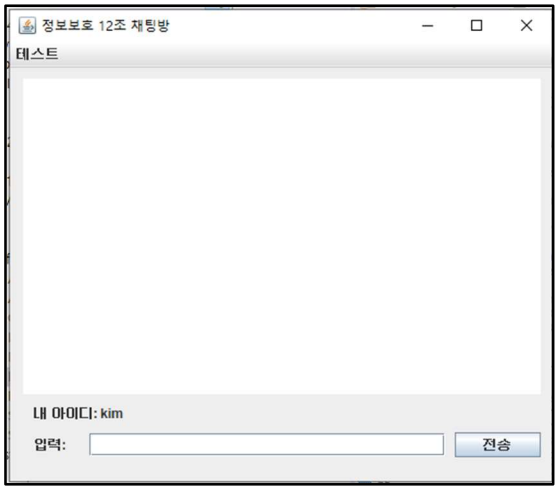
아래의 소스는 Server의 run() method의 일부를 나타낸 것이다. 서버는 LoginRequest의 tag를 보고, DBManager를 통해 데이터베이스를 확인한다. 해당 입력값들이 데이터베이스의 유저 정보와 일치한다면, 로그인을 진행하고 채팅 화면을 보여준다. Server는 입력받은 암호를 SHA-256 알고리즘을 이용하여 암호화한 후, 값을 비교한다.

```
if(type.equals("LoginRequest")){
    String id=st.nextToken().toString();
    String password= DBManager.get_salt_from_DB(id)+st.nextToken().toString();
    password = SHACrypto.sha256(password);
    int result=DBManager.get_table_from_DB(id, password);
    if(result>=1){
        //login시 메세지를 서버가 클라이언트로 보냅니다.
        System.out.println(date_time+"["+type+"]"+"Client "+id +"'s RSA password : "+password);
        sendMsg_login("success");
        continue;
    }else{
        System.out.println(date_time+"Database login fail");
        //실패시 action
        sendMsg_login("fail");
        continue;
    }
}
```

[그림 3-2-2] Client login UI



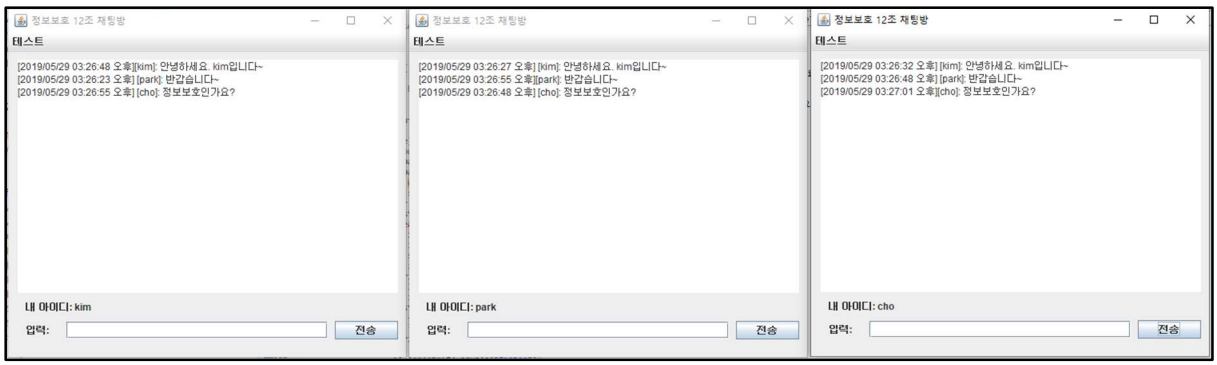
[그림 3-2-3] Client login UI



[그림 3-2-4] Client의 Chatting UI

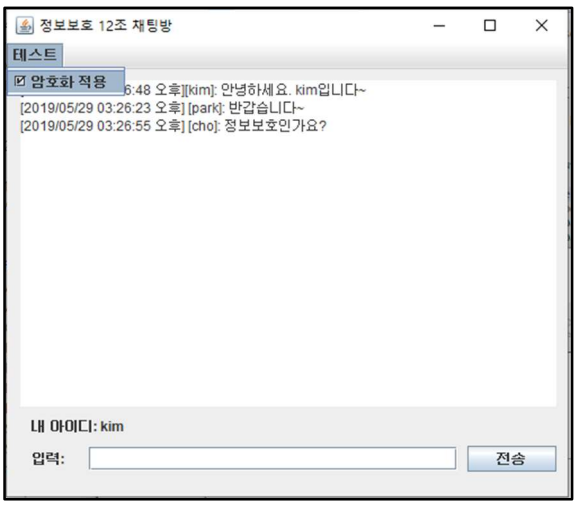
- 채팅 서비스

여러 사용자가 로그인에 성공한다면, 채팅 서비스에 접근할 수 있다. 각각의 사용자들은 하단의 필드를 통해 보내고 싶은 메시지를 전송한다. 보낸 메시지는 다른 사용자의 상단 필드에 표시된다. 사용자는 자신이 보낸 메시지와 다른 사용자가 보낸 메시지를 확인할 수 있으며, 각각의 내용은 시간과 보낸 사용자의 id와 함께 필드에 표시된다.

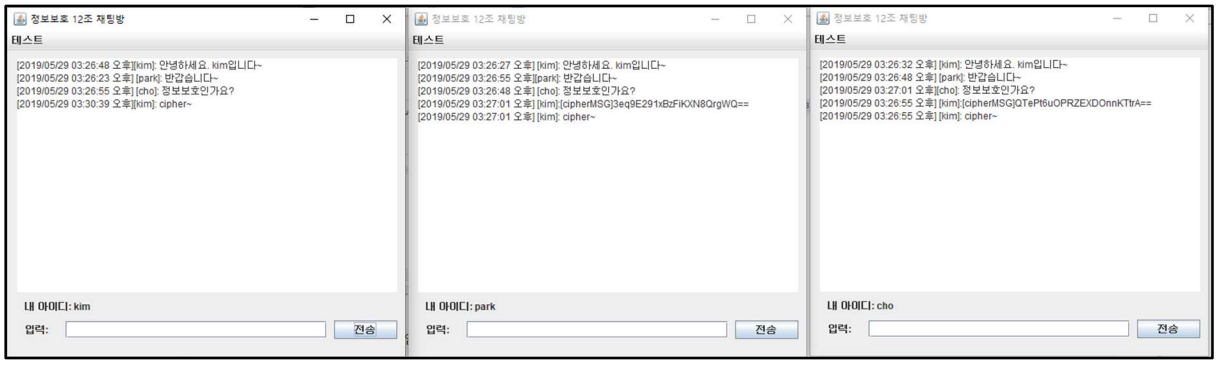


[그림 3-2-5] 사용자간 채팅 서비스 예제

만약 특정 사용자가 암호화된 메시지를 보내고 싶은 경우, 사용자는 메뉴의 테스트에서 암호화 적용이라는 체크박스를 선택한다. 암호화 적용 체크박스가 적용된 후, 사용자가 평문의 메시지를 보낸다면 사용자들은 서버가 보내준 암호화 메시지를 확인 후 평문으로 변환한다.



[그림 3-2-6] 사용자가 암호화 메시지를 전송하는 경우



[그림 3-2-7] 특정 사용자가 암호화 메시지를 전송시 다른 사용자가 받은 메시지

사용자들이 로그인을 하는 시점부터 각 메시지를 전송하는 과정은 서버의 console로 확인이 가능하다. 사용자가 로그인을 할 시, 테스트 목적으로 각각의 사용자의 암호가 서버의 console에 출력된다. 또한 각 사용자들이 보내는 평문 메시지와 암호화된 메시지 또한 console에 출력된다. 특히 사용자가 암호화를 선택한 후 메시지를 보내면, 메시지를 보낸 사용자가 암호화한 암호화 메시지가 서버로 전송되고, 서버는 이를 console에 출력 후 평문으로 바꾼다. 평문으로 바꾼 메시지를 서버가 다시 암호화해 다른 사용자에게 보내면 각 사용자들은 암호화된 메시지를 평문 메시지로 해독한다.

```
result: 1
[2019/05/29 03:26:23 오후] [LoginRequest]Client kim's     password : 7cc9311cfb06a3b09c96f1fc1f32bddca2a83ad53407e05822e8c8933ce87e02
result: 2
[2019/05/29 03:26:27 오후] [LoginRequest]Client park's     password : 542b048cbbf48d586c7af3d307160a953bbce3b06977680ecf83e281f1649461
result: 3
[2019/05/29 03:26:32 오후] [LoginRequest]Client cho's     password : e14778a55f77741fdb10275c0feb9f8899b2a2c039971c8929936dbe774cd446
[2019/05/29 03:26:48 오후] [Message][kim]: 안녕하세요. kim입니다~
[2019/05/29 03:26:55 오후] [Message][park]: 반갑습니다~
[2019/05/29 03:27:01 오후] [Message][cho]: 정보보호인가요?
[2019/05/29 03:30:39 오후] [Cipher][kim]: YACOeFj+O2MLxR3rUMXN4g==
```

[그림 3-2-8] 서버에 기록되는 로그 출력

3) 암호화

- SHA-256

```
public class SHACrypto {

    public static String sha256(String msg)  throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        md.update(msg.getBytes());
        return byteToHexString(md.digest());
    }

    public static String byteToHexString(byte[] data) {
        StringBuilder builder = new StringBuilder();
        for(byte b : data) {
            builder.append(String.format("%02x", b));
        }
        return builder.toString();
    }

}
```

[그림 3-3-1] SHACrypto.java

프로그램에서 Password를 암호화하기 위해 사용하였으며, 암호화의 강도를 높이기 위해 DB에서 입력이 들어온 ID에 해당하는 salt를 가져와 Password와 조합한 뒤 Hashing을 진행해 DB에 저장되어있는 Password와 일치하는지 확인한다. 즉, DB에는 Password의 평문 형태는 저장되어있지 않으며, 여러 유저가 같은 암호를 사용하더라도 유저마다 고유한 salt를 추가한 뒤 Hashing을 진행해 Attacker가 암호를 쉽게 알지 못하게 구현하였다.

sha256(String msg): msg를 SHA-256을 통해 Hasing하여 결과를 return하는 함수.

- AES-128

```
public AESCrypto(String key) throws UnsupportedEncodingException {
    this.iv = key.substring(0, 16);

    byte[] keyBytes = new byte[16];
    byte[] b = key.getBytes("UTF-8");
    int len = b.length;
    if (len > keyBytes.length) {
        len = keyBytes.length;
    }
    System.arraycopy(b, 0, keyBytes, 0, len);
    SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");

    this.keySpec = keySpec;
}

public String aesEncode(String plainText) throws java.io.UnsupportedEncodingException, NoSuchAlgorithmException
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, keySpec, new IvParameterSpec(iv.getBytes()));

    byte[] encrypted = cipher.doFinal(plainText.getBytes("UTF-8"));
    String enStr = new String(Base64.encodeBase64(encrypted));

    return enStr;
}

public String aesDecode(String cryptText) throws java.io.UnsupportedEncodingException, NoSuchAlgorithmException
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.DECRYPT_MODE, keySpec, new IvParameterSpec(iv.getBytes("UTF-8")));

    byte[] byteStr = Base64.decodeBase64(cryptText.getBytes());
    String deStr = new String(cipher.doFinal(byteStr),"UTF-8");

    return deStr;
}
```

[그림 3-3-2] AESCrypto.java

프로그램에서 Message를 암호화하기 위해 사용하였으며, 후술할 RSA를 통해 서버가 클라이언트로부터 받은 AES 공개키를 통해 서로 Message의 암복호화를 진행한다.

AESCrypto(String key): AES 암호화 알고리즘에 사용할 키를 만든다.

aesEncode(String plainText): plainText를 publicKey를 통해 암호화하는 함수.

aesDecode(String cryptText): cipherText를 privateKey를 통해 복호화하는 함수.

- RSA-2048

```
public RSACrypto(int keyBits) throws NoSuchAlgorithmException {
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(keyBits);

    KeyPair keyPair = keyPairGenerator.genKeyPair();

    this.publicKey = keyPair.getPublic();
    this.privateKey = keyPair.getPrivate();
}

public static String encrypt(Key publicKey, String plainText) throws NoSuchAlgorithmException,
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.ENCRYPT_MODE, publicKey);

    byte[] encrypted = new byte[2048];
    encrypted = cipher.doFinal(plainText.getBytes("UTF-8"));
    String enStr = new String(Base64.encodeBase64(encrypted));

    return enStr;
}

public static String decrypt(Key privateKey, String cryptText) throws NoSuchAlgorithmException,
    Cipher cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
    cipher.init(Cipher.DECRYPT_MODE, privateKey);

    byte[] byteStr = Base64.decodeBase64(cryptText.getBytes());
    String deStr = new String(cipher.doFinal(byteStr),"UTF-8");

    return deStr;
}
```

[그림 3-3-3] RSACrypto.java

프로그램에서 AES 공개키를 암호화하기 위해 사용하였다. 서버측에서 RSA 공개키를 클라이언트에게 전송하고 클라이언트는 RSA 공개키를 통해 AES 공개키를 암호화하여 서버에게 전송한 뒤, 서버는 이를 RSA 개인키를 통해 복호화하여 AES 공개키를 취득하는 방식으로 구현하였다.

RSACrypto(int keyBits): RSA 암호화 알고리즘에 사용할 키쌍(공개키, 개인키)을 만든다.

encrypt(Key publicKey, String plainText): plainText를 publicKey를 통해 암호화하는 함수.

decrypt(Key privateKey, String cryptText): cipherText를 privateKey를 통해 복호화하는 함수.

4. 결론

여러 암호화 알고리즘을 통하여 간단한 보안 프로그램을 구현해보았다. 시중에 서비스되고 있는 "텔레그램"의 암호화 방식을 채택하여 구현을 하려 했으나, n대n 채팅 방식인 이 프로그램과는 달리 텔레그램은 1대1 채팅 방식이고, 이 때문에 기존과는 다른 암호화 방식이 요구되었다. 이 때문에 유저마다 서로 다른 키를 가지고 있어 번거롭고 서버에서 메시지를 볼 수 있다는 문제점이 있었다. 이를 해결하기 위해서는 모든 클라이언트가 같은 AES 키를 가질 수 있게 하는 방법이 필요할 것으로 보인다.

인류의 암호화의 역사는 고대 전쟁 때부터 글을 숨기려고 할 때부터 시작되어왔다. 현대에 들어서는 전산학과 그 부속품들이 급격하게 발달함에 따라, 암호화할 대상은 글에서부터 파일, 사진, 동영상, 음성 등 다수의 대상으로 확장되어왔고, 과거의 방식들은 더 이상 안전한 것들이 아니게 됐다. 아마 인류가 사라지지 않는 한, 남들에게 보여야하지 말아야 할 무언가는 계속 존재할 것이다. 암호화 방식과 암호화를 뚫는 방식은 끊임없이 발달할 것이고, 암호화할 무언가는 더욱 더 많아질 것이다. 이는, 우리가 나중에 사회에 나갔을 때에도 변하지 않으며 이번 프로젝트를 진행하면서 얻은 지식과 경험은 먼 미래에도 변함없이 쓰일 가치 있는 것이라고 생각한다.

5. 그림 출처

[그림 1-1-1]: https://www.boannews.com/media/view.asp?idx=40472

[그림 1-1-2]: https://daumdam.tistory.com/288

[그림 1-2-1]: https://samsclass.info/123/proj10/p3-sniff.htm

[그림 2-1-1]: http://www.bitweb.co.kr/news/view.php?idx=388

[그림 2-2-1]: https://pequalsnp-team.github.io/writeups/Bad-Aes

[그림 2-2-2]: https://de.wikipedia.org/wiki/Rijndael_MixColumns