GR 8

Secure Instant Messaging

Chenguang Zhang
Ziqi Chen
Lingqi Meng

Context

- Motive
- Our Work
- Implementation Details
- References

Motive

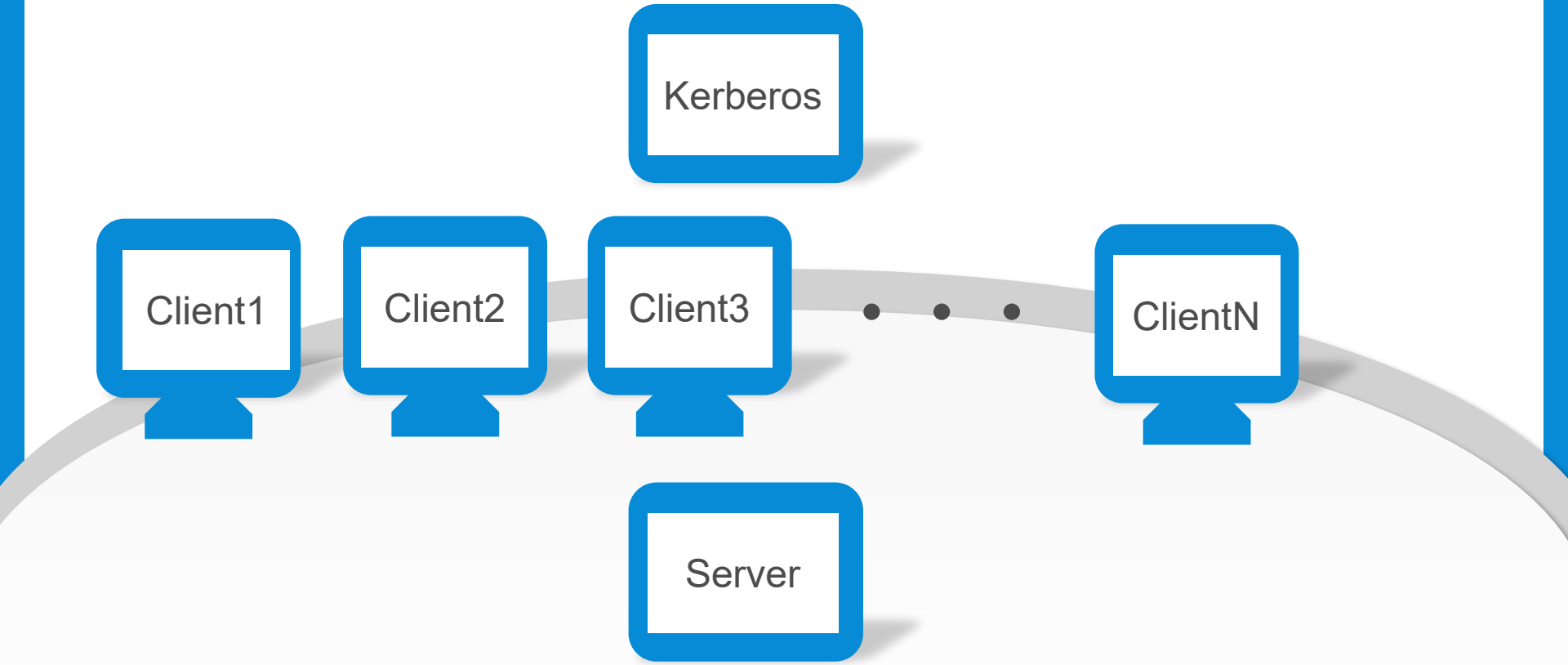
Security Challenges

- Replay
- Eavesdropping
- Replay
- …

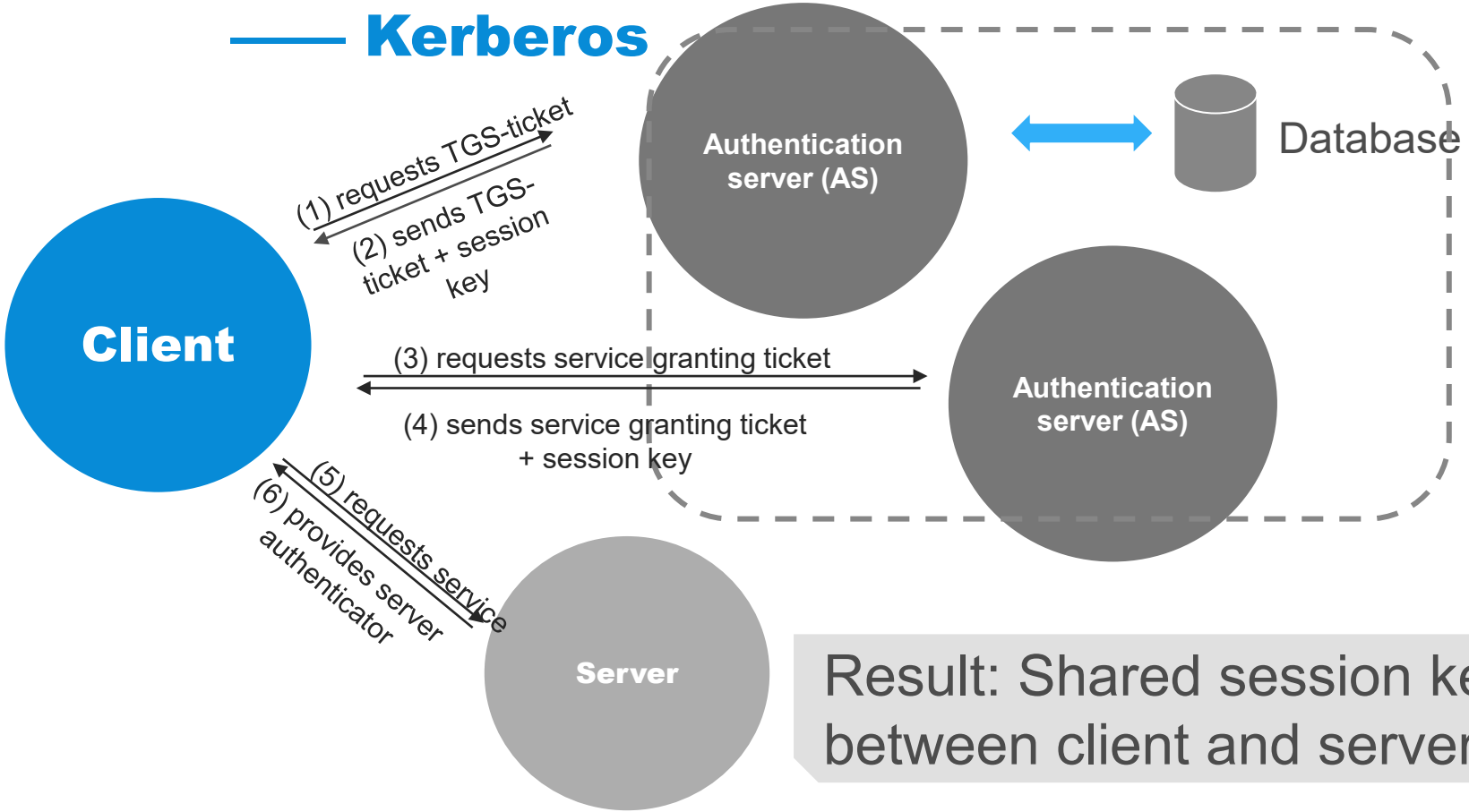
Our work

1. A secure instant messaging system
2. Functionality: logging in, online list, secure text messaging, etc.
3. Security Services: confidentiality, peer-entity and message authentication, integrity, nonrepudiation

Client-Server Architecture with Kerberos Authentication



Phase I : Authentication —— Kerberos

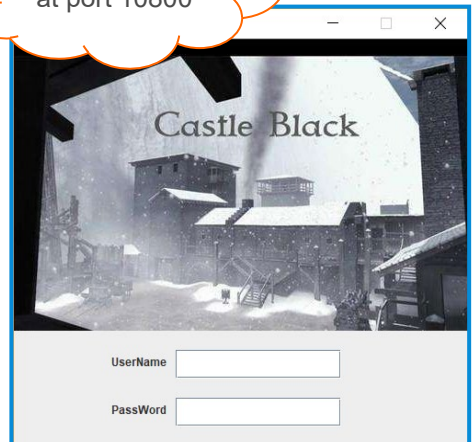


Result: Shared session key between client and server

Phase I


```
public AS() {
    sessiongeneration aeskey = new sessiongeneration();
    sessionkeytgsclient = aeskey.aeskey();
    begintime = a.begintime();
    endtime = a.endtime();
    try {
        ASsocket = new ServerSocket(10800);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

AS listening at port 10800




```
String transmit = clientid + "@" + tgsid + "@" + begintime;
```

04fc711301f3c784d66955d98d399afb@11111111@20171212200526


```
if (tgsidfromsocket.equals(tgsid) && clientIdflag) {
    String tickettgsunprocess = sessionkeytgsclient + "@" + clientidfromsocket + "@" + tgsid;
    String tickettgs = AES.Encrypt(tickettgsunprocess, tgskey);
    String resultunprocess = sessionkeytgsclient + "@" + tgsid + "@" + tickettgs;
    result = AES.Encrypt(resultunprocess, clientkey);
    System.out.println("the TGS ticket in AS is " + tickettgs);
} else {
    result = "unauthorized client";
}
```

suTguRN2FjschXw=@11111111@
4f15e903add2d7b557613ea42588
53b0255e3fe4442166b59d692cd8c
969ca58fffaf45e56109b649a4cb2d
b78f13c9e35cd8b565ba938435203
67282e30cbf4

Request TGS

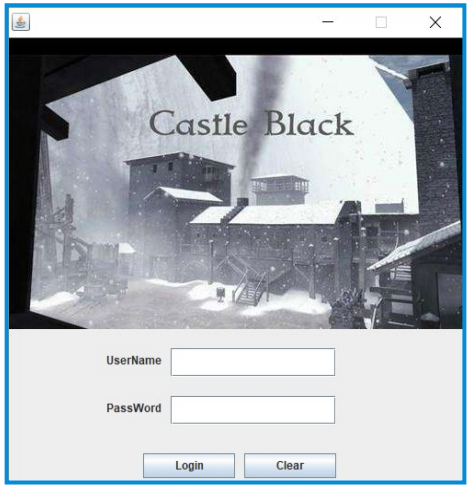
Client → TGS

```
public TGS() {
    sessiongeneration aeskey = new sessiongeneration();
    sessionkeyclientserver = aeskey.aeskey();
    try {
        TGSsocket = new ServerSocket(10801);
    } catch (Exception e) {
        e.printStackTrace();
    }
```

TGS listening at port 10801

```
String auth = AES.Encrypt(clientid, sessionkeytgsclient);
String transmit = serverid + "@" + tickettgs + "@" + auth;
```

222222222@4f15e903add2d7b557613ea4258853b0255e3fe4442166b59d692cd8c969ca58fffaf45e56109b649a4cb2db78f13c9e35cd8b565ba93843520367282e30cbf4@2d6a80c862c1039de8bc9d21a2251b25

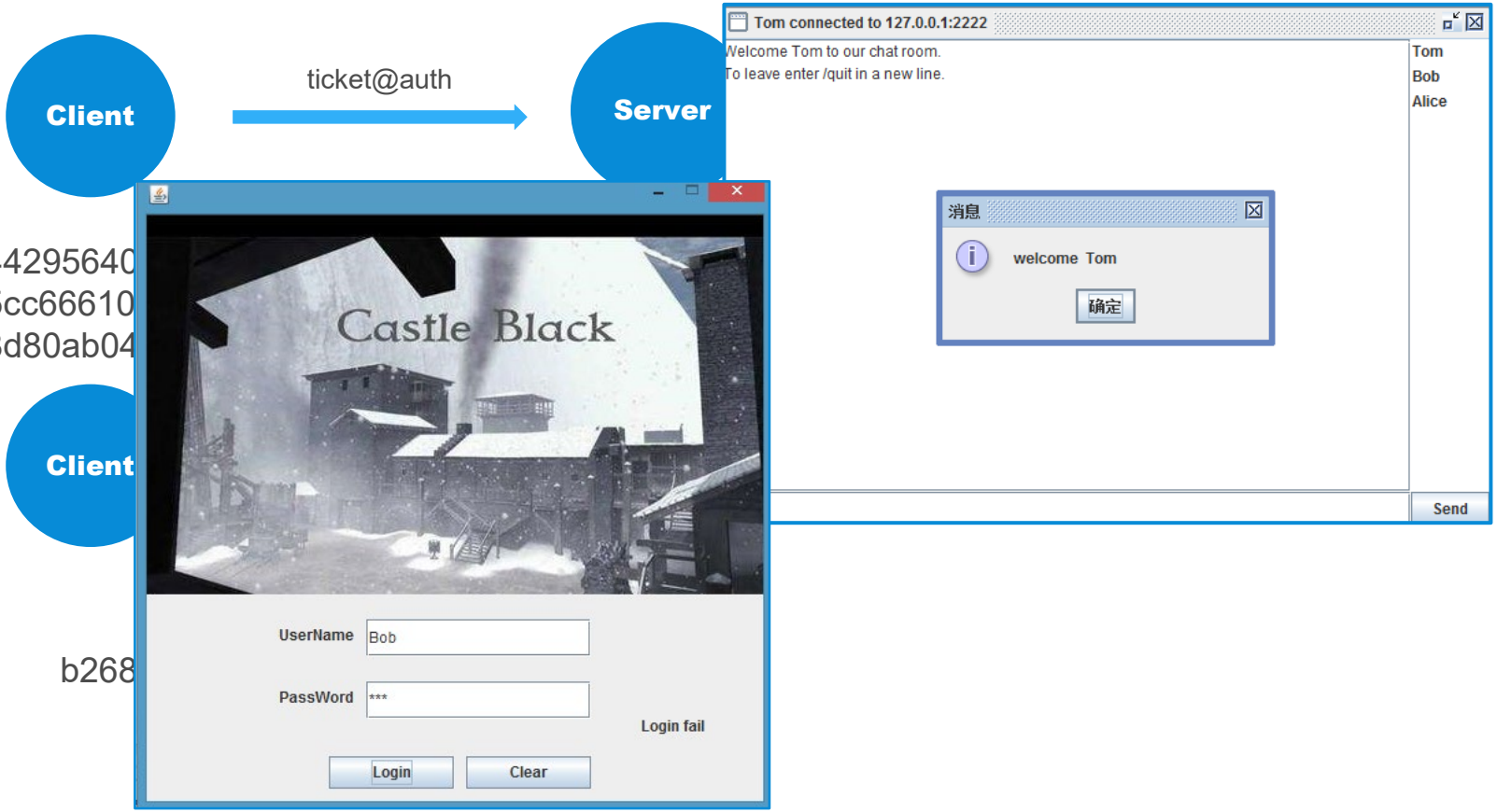


Client ← TGS

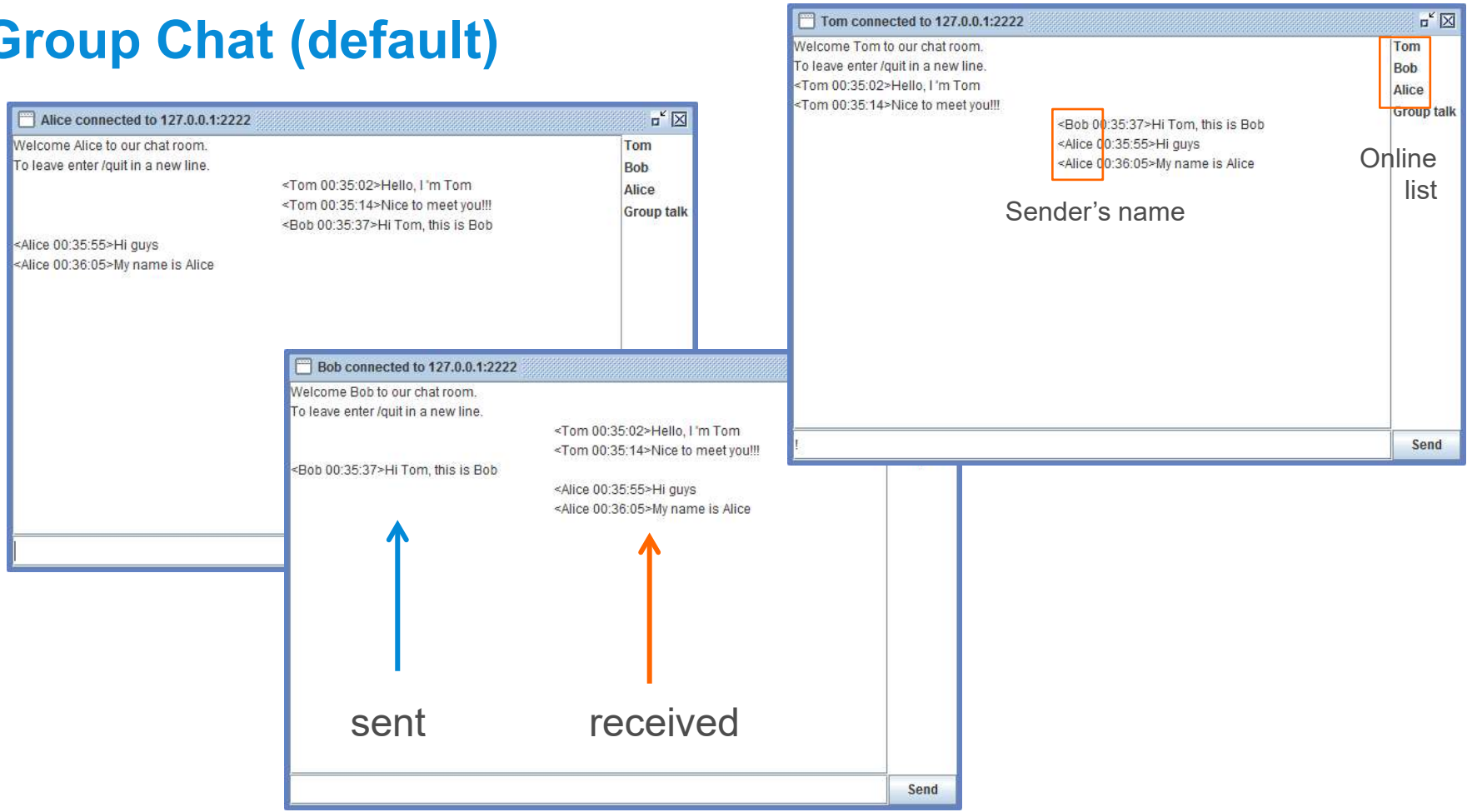
ofBtvBWdGZ3LDio=@222222222@442956401baf0a28359cd224f54ccfbf395df880e32f65cc66610139d98b3ba4ed76e6e2f9c1003d26117d6d8d80ab04

```
if(clientidfromAS.equals(clientidfromclient)&&TGSidfromAS.equals(tgsid)&&serveridfromclient.equals(serverid)){
    String ticketserverunprocess = sessionkeyclientserver + "@" + clientidfromclient + "@" + serverid;
    String ticketserver = AES.Encrypt(ticketserverunprocess, serverkey);
    String transmitunprocess = sessionkeyclientserver + "@" + serverid + "@" + ticketserver;
    result = AES.Encrypt(transmitunprocess, sessionkeytgsclient);
}else{
    result = "unauthorized client";
}
```

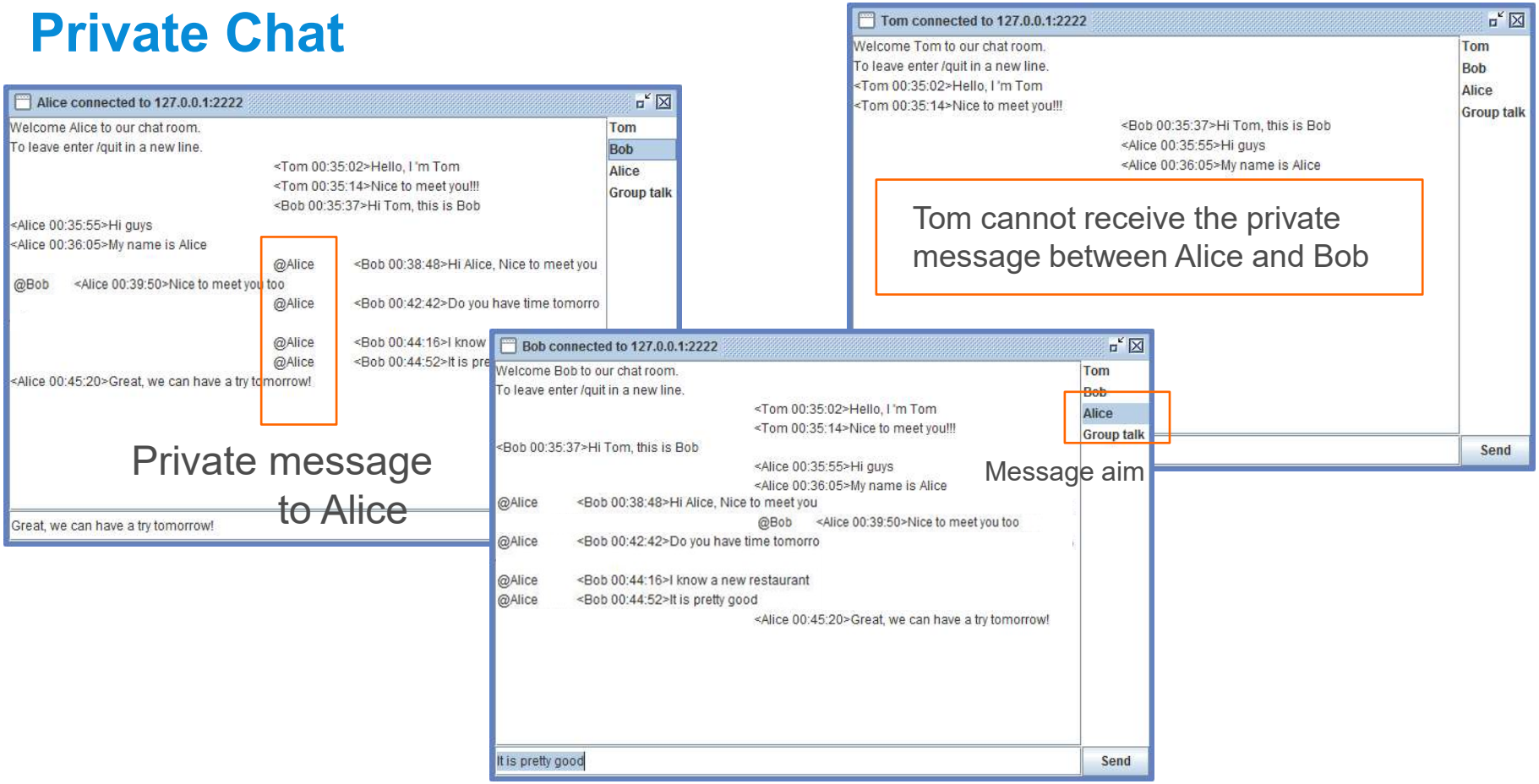
Connect to server



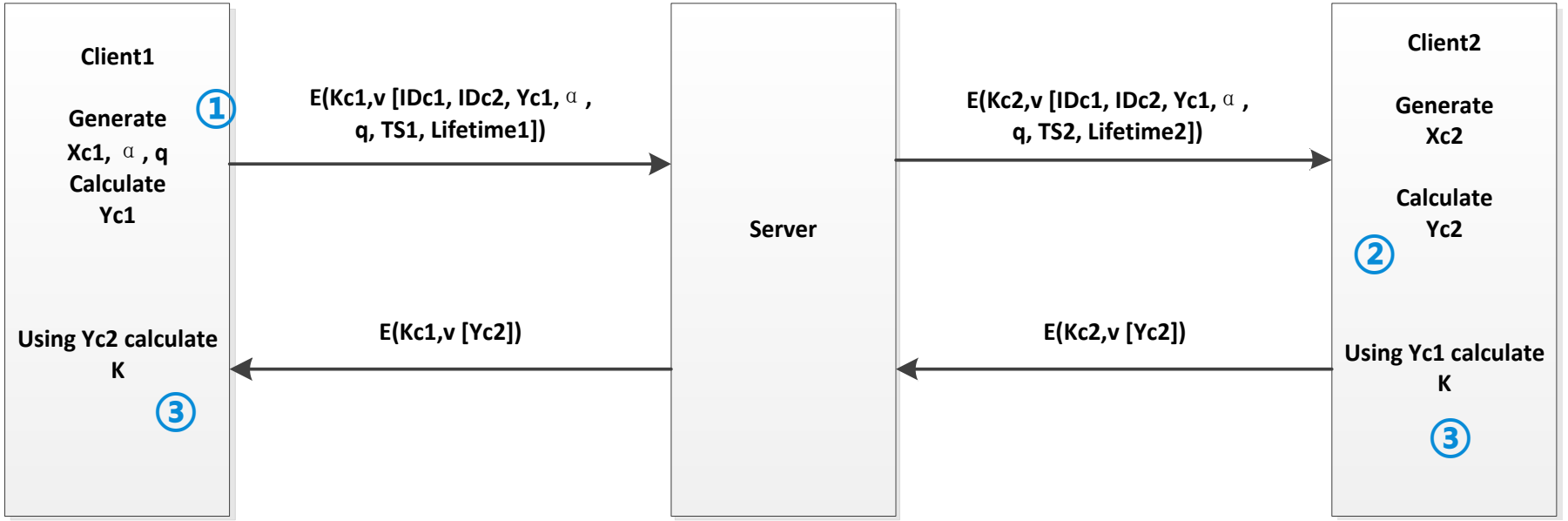
Group Chat (default)



Private Chat



Phase II : Client Key Exchange
—— Diffie-Hellman Algorithm



Result: Exchange secret key K between Client1 and Client2

Phase II : Client Key Exchange
—— Diffie-Hellman Algorithm

①

```
public HQKeyPair initPartyAKey(int keySize) throws NoSuchAlgorithmException
{
    KeyPairGenerator keyPairGen = KeyPairGener
    keyPairGen.initialize(1024);
    KeyPair keyPair = keyPairGen.generateKeyPa
    return new HQKeyPair(keyPair);
}
```

②

```
public HQKeyPair initPartyBKey(byte[] partyAPublicKey) throws Exception
{
    X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(partyAPublicKey);
    KeyFactory keyFactory = KeyFactory.getInstance(ALGORITHM);
    PublicKey pubKey = keyFactory.generatePublic(x509KeySpec);

    DHParameterSpec dhParamSpec = ((DHPublicKey) pubKey).getParams();

    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance(keyFactory.getAlgorithm());
```

③

```
private SecretKey getSecretKey(byte[] publicKey, byte[] privateKey, String algorithm) throws Exception
{
    KeyFactory keyFactory = KeyFactory.getInstance(ALGORITHM);
    X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(publicKey);
    PublicKey pubKey = keyFactory.generatePublic(x509KeySpec);

    PKCS8EncodedKeySpec pkcs
    Key priKey = keyFactory.

    KeyAgreement keyAgree =
    keyAgree.init(priKey);
    keyAgree.doPhase(pubKey,

    SecretKey secretKey = ke

    return secretKey;
}
```

Problems | Javadoc | Declaration | Console | LogCat

```
login (1) [Java Application] D:\softwear\JDK\bin\javaw.exe (2017-12-14 上午4:16:25)
B_PublicKey: MIIBpjCCARsGCSqGSIb3DQEDATCCAQwCgYEA/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuAHTRv8mZgt2uZUKWk
A_PrivateKey: MIIBZwIBADCCARsGCSqGSIb3DQEDATCCAQwCgYEA/X9TgR11EilS30qcLuzk5/YRt1I870QAwx4/gLZRJmlFXUAiUftZPY1Y+r/F9bow9subVWzXgTuAHTRv8mZgt2u
secretKey: EDxcpK7f6hy5JvQ=
========================================
before encrypte: <Bob 04:16:55>Hello everyone
after encrypte: a5d047b7a246c2033084630ed8a581fa8d04ac0dedfe0cca00669eac7f220cdd
========================================
before_decrypte: a5d047b7a246c2033084630ed8a581fa8d04ac0dedfe0cca00669eac7f220cdd
after_decrypte: <Bob 04:16:55>Hello everyone
========================================
```

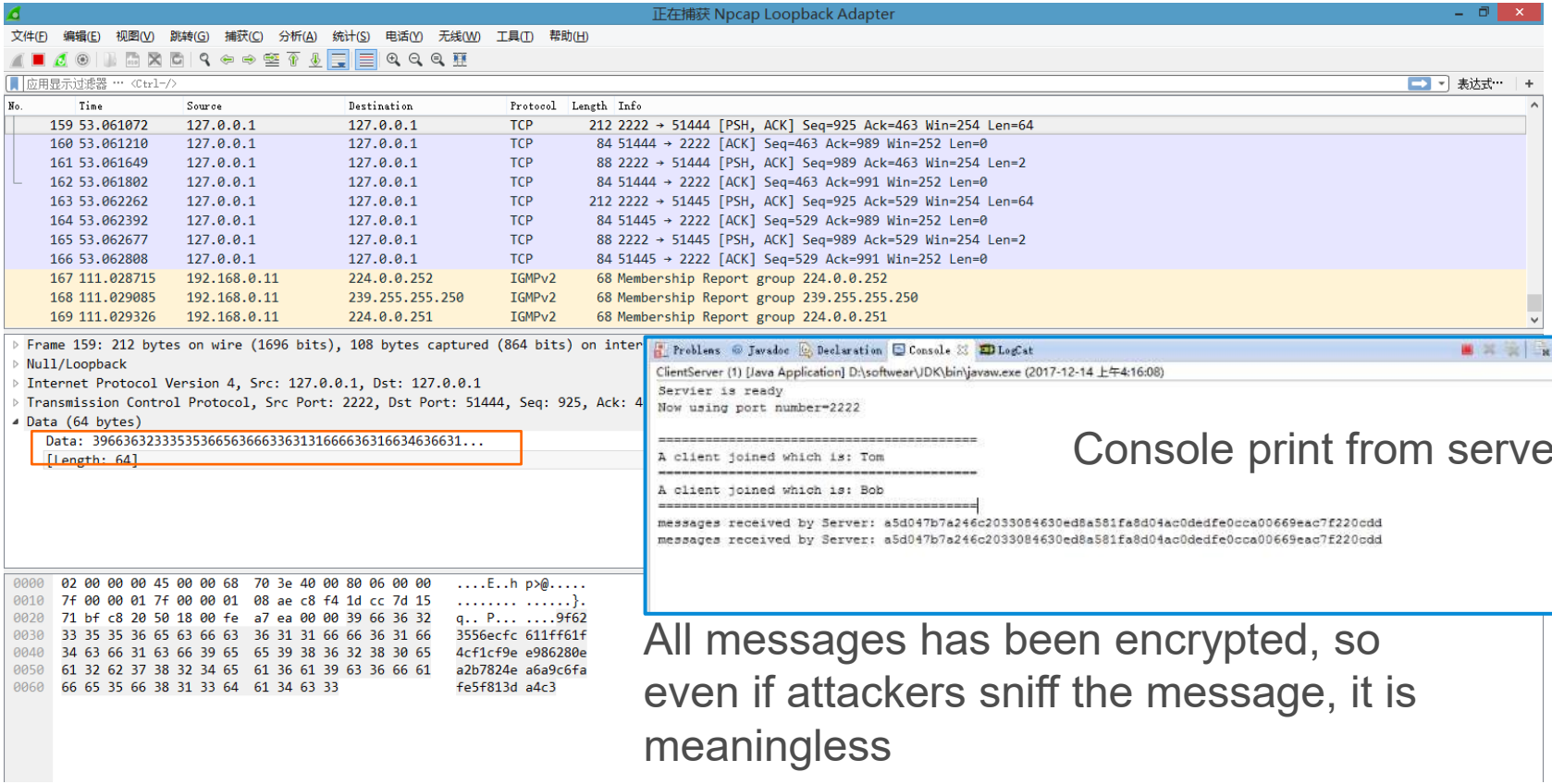
Console print from client

Phase III : Encrypted Communication —— AES

```java
// encryption
public static String Encrypt(String plaintext, String sKey) throws Exception {
    if (sKey == null) {
        System.out.println("Key cannot be null");
        return null;
    }

    if (sKey.length() != 16) {
        System.out.println("Key's length must be 16");
        return null;
    }

    byte[] raw = sKey.getBytes();
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    IvParameterSpec iv = new IvParameterSpec("0102030405060708".getBytes());
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec, iv);
    byte[] encrypted = cipher.doFinal(plaintext.getBytes());

    return byte2hex(encrypted).toLowerCase();
}
```

```java
public static String Decrypt(String myCipher, String sKey) throws Exception {
    try {

        if (sKey == null) {
            System.out.println("Key cannot be null");
            return null;
        }

        if (sKey.length() != 16) {
            System.out.println("Key's length must be 16");
            return null;
        }

        byte[] raw = sKey.getBytes("ASCII");
        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        IvParameterSpec iv = new IvParameterSpec("0102030405060708".getBytes());
        cipher.init(Cipher.DECRYPT_MODE, skeySpec, iv);
        byte[] encrypted1 = hex2byte(myCipher);

        try {
            byte[] original = cipher.doFinal(encrypted1);
            String originalString = new String(original);
            return originalString;
        } catch (Exception e) {
            System.out.println(e.toString());
            return null;
        }

            } catch (Exception ex) {
                System.out.println(ex.toString());
                return null;
            }
        }
```

Bad Test --Sniffing



Reference

1. Hiroaki Kikuchi, Minako Tada. (2004) Secure Instant Messaging Protocol Preserving Confidentiality against Administrator
2. William Stallings. (2011) Network Security Essentials: Applications and Standards, Fourth Edition.
3. Jimmy Florez Z, Camilo Logreira R. (2016) Architecture of Instant Messaging Systems for Secure Data Transmission
4. Changji Wang, Welong Lin (2013) Design of An Instant Messaging System Using Identity Based Cryptosystems.
5. B.Clifford Neuman, Theodore Ts'o. (1994) Kerberos: An Authentication Service for Computer Networks.
6. Anita Narwal, Sunita Tomar. (2015) Kerberos Protocol: A Review
7. Michael Steiner, Gene Tsudik. (1996) Diffie-Hellman Key Distribution Extended to Group Communication.

Thank you

Chenguang Zhang
Ziqi Chen
Lingqi Meng