Documentação Chat Criptografado

```
RSACryptoServiceProvider RSAprovider = new RSACryptoServiceProvider();
RSAParameters publicKey, privateKey;
DESCryptoServiceProvider DESprovider = new DESCryptoServiceProvider();
byte[] Key, IV;
```

Linhas 48-51 do arquivo SocketConnection.cs, aqui é responsável por instanciar um provider para os algoritmos RSA e DES. No DES a chave consiste de um vetor de inicialização (IV - em byte[]) e chave propriamente dita (Key).

```
private SocketConnection()
{
    token = tokenSource.Token;
    publicKey = RSAprovider.ExportParameters(false);
    privateKey = RSAprovider.ExportParameters(true);
    Key = DESprovider.Key;
    IV = DESprovider.IV;
```

No mesmo arquivo, aqui capturamos a chave pública e privada do RSA para depois usála. ExportParameters(false) captura a chave pública e ExportParameters(true) captura a chave privada.

No DES a chave é pega através dos atributos Key e IV.

```
302
303             byte[] encryptedKey = new byte[1], encryptedIV = new byte[1];
304             UserChat owner;
305             RSACryptoServiceProvider ownerProvider = new RSACryptoServiceProvider();
306             if (room is UserRoom)
307             {
308                 owner = Users.getInstance().UserCollection.First(s => s.Id == room.Id);
309                 ownerProvider.ImportCspBlob(owner.PUBLIC_KEY);
310                 encryptedMessage = Encrypt(message, Key, IV);
311                 encryptedKey = ownerProvider.Encrypt(Key, false);
312                 encryptedIV = ownerProvider.Encrypt(IV, false);
313
314                 dynamic _message = new
315                 {
316                     id = room.Id,
317                     isGroup = room is GroupRoom,
318                     message = encryptedMessage,
319                     key = Convert.ToBase64String(encryptedKey),
320                     iv = Convert.ToBase64String(encryptedIV),
321                     senderId = User.getInstance().Id,
322                     command = DispatcherCodes.SEND
323                 };
324
325                 send(serializer.Serialize(_message));
326                 sendDone.WaitOne();
```

Quando host A e B irão se comunicar,a linha 309 importa a chave pública de B e encripta a chave de sessão de A (DES), em seguida o método Encrypt (ln 310) é chamado para encriptar a mensagem com a chave de sessão de A e o vetor de inicialização.
Em seguida, a chave de sessão (DES) é criptografada e o vetor também.
Linha 313 cria um payload (carga de trabalho) para enviar via socket com a chave e o vetor e a mensagem criptografada.

```
dynamic message = new
{
    id = user.ID,
    sender = sendto.ID,
    command = DispatcherCodes.SEND,
    message = data["message"],
    key = data["key"],
    iv = data["iv"]
};
send(serializer.Serialize(message), sendto.Socket);
sendDone.WaitOne();
send(serializer.Serialize(message), client);
sendDone.WaitOne();
```

Já no servidor, a mensagem só é repassada para o outro lado (host B) com a chave de sessão criptografada de A e a mensagem. Na outra ponto host B decripta com a chave privada. Se B quiser mandar uma mensagem deverá fazer o mesmo processo.

```
532             {
533                 user = Users.getInstance().UserCollection.FirstOrDefault(s => s.Id == json["id"]);
534                 RSACryptoServiceProvider decryptor = new RSACryptoServiceProvider();
535                 decryptor.ImportCspBlob(RSAprovider.ExportCspBlob(true));
536
537                 byte[] encryptedKey = Convert.FromBase64String(json["key"]);
538                 byte[] encryptedIV = Convert.FromBase64String(json["iv"]);
539
540                 byte[] decryptedKey = decryptor.Decrypt(encryptedKey, false);
541                 byte[] decryptedIV = decryptor.Decrypt(encryptedIV, false);
542
543                 string decryptedMessage = Decrypt(json["message"], decryptedKey, decryptedIV);
544
545                 message = String.Format("{0} disse: {1}", user.Alias, decryptedMessage);
```

Linha 534 cria um decryptor com a chave privada do host B. Decripta o vetor de inicialização e chave de sessão de A. A função Decrypt, decripta dados usando a chave privada do RSA. (linha 540 e 541).

A função Decrypt e Encrypt, decripta e encripta mensagens no DES respectivamente.

```
public string Encrypt(string originalString, byte[] KEY, byte[] IV)
{
    DESCryptoServiceProvider cryptoProvider = new DESCryptoServiceProvider();
    MemoryStream memoryStream = new MemoryStream();
    CryptoStream cryptoStream = new CryptoStream(memoryStream,
        cryptoProvider.CreateEncryptor(KEY, IV), CryptoStreamMode.Write);
    StreamWriter writer = new StreamWriter(cryptoStream);
    writer.Write(originalString);
    writer.Flush();
    cryptoStream.FlushFinalBlock();
    return Convert.ToBase64String(memoryStream.GetBuffer(), 0, (int)memoryStream.Length);
}

public string Decrypt(string cryptedString, byte[] KEY, byte[] IV)
{
    DESCryptoServiceProvider cryptoProvider = new DESCryptoServiceProvider();
    MemoryStream memoryStream = new MemoryStream
            (Convert.FromBase64String(cryptedString));
    CryptoStream cryptoStream = new CryptoStream(memoryStream,
        cryptoProvider.CreateDecryptor(KEY, IV), CryptoStreamMode.Read);
    StreamReader reader = new StreamReader(cryptoStream);
    return reader.ReadToEnd();
}
```

Cria-se um provider para o DES, um fluxo de memória (MemoryStream). e uma classe auxiliar para tratar um fluxo criptografado (CryptoStream). Passa-se como parâmetro a Key e IV e o modo que se quer ler ou escrever (CryptoStreamMode.Write ou CryptoStreamMode.Read. Um objeto StreamWriter é criado para escrever a informação original. Importante fazer o flush do bloco final, caso ainda tenha algo em memória. Uma string em base 64 é retornada.

A função de decriptar faz o processo oposto. É mais simples,basta informar o modo (CryptoStreamMode.Read) e ler o stream até o final ReadToEnd().

Quando a comunicação é feito para um grupo (uma sala de bate-papo) a sala possui uma chave RSA própria e uma chave de sessão própria para fazer a comunicação.

```
public class Group
{
    public String ID { get; set; }
    public List<User> Subscribers { get; }
    public String Alias { get; set; }
    public User leader { get; set; }

    public RSACryptoServiceProvider rsa { get; set; }
    public DESCryptoServiceProvider des { get; set; }

    public Group()
    {
        this.Subscribers = new List<User>();
    }
}
```

Logo no início o servidor Encripta a chave de sessão e o vetor da sala usando a chave pública dos usuários que fazem parte dela, e depois as envia. Do outro lado,o cliente decripta a chave e o vetor da sala usando sua chave privada e armazena em local seguro.

O cliente quando quer se comunicar envia apenas a mensagem,não precisando enviar nenhuma outra chave, já que ele guarda a chave de sessão da sala em local seguro, para o servidor. O servidor então repassa, o outro lado como também faz parte da sala e guarda essas informações,apenas decripta a mensagem utilizando a chave e vetor da sala.

```
if(room is GroupRoom)
{
    user = Users.getInstance().UserCollection.FirstOrDefault(s => s.Id == json["sender"]);
    if (user == null)
    {
        message = String.Format("{0} disse: {1}", "Você", Decrypt(json["message"],
            (room as GroupRoom).Key, (room as GroupRoom).IV));
    }
    else
    {
        message = String.Format("{0} disse: {1}", user.Alias, Decrypt(json["message"],
            (room as GroupRoom).Key, (room as GroupRoom).IV));

    }
```

```
public class GroupRoom : Room
{
    public GroupRoom()
    {
        Subscribers = new ObservableCollection<UserChat>();
    }

    public byte[] PUBLIC_KEY;
    public byte[] Key, IV;

    public UserChat Leader { get; set; }
    public ObservableCollection<UserChat> Subscribers { get; set; }
}
```