Desenvolvimento de um chat seguro utilizando a metodologia Diffie-Hellman

Anna Carolina Kern Gonçalves, Gabriel Longhi Quadro, Léo Henrique Eifert, Rogus Staub, Wiliam Kern Franco

Departamento de Computação – Universidade de Santa Cruz do Sul (UNISC)
96.815-900 – Santa Cruz do Sul – RS – Brazil

{annagoncalves, gabrielquadro, leoeifert, rogus, wkern}@mx2.unisc.br

Abstract. *The work developed is for the discipline of Security in Information Technology, in this report the step by step development of a Secure Chat with the Diffie-Hellman methodology to perform encryption using the AES algorithm is demonstrated, in this chat the user will be able to choose the size of the encryption key and you can view the sent/received message encrypted and decrypted.*

Resumo. *O trabalho desenvolvido é para a disciplina de Segurança em Tecnologia da Informação, neste relatório são demonstrado o passo a passo do desenvolvimento de um Chat Seguro com a metodologia Diffie-Hellman para realizar a criptografia usando o algoritmo AES, neste chat o usuário poderá escolher o tamanho da chave de criptografia e poderá visualizar a mensagem enviar/recebida criptografada e descriptografada.*

1. Introdução

O trabalho relatado neste relatório foi desenvolvido para a disciplina de Segurança em Tecnologia da Informação. Este relatório consiste na descrição do desenvolvimento de um Chat Seguro utilizando criptografia *AES* com a metodologia *Diffie-Hellman.*

Neste *chat* o usuário poderá realizar a configuração escolhendo o tamanho da chave que será utilizada para criptografar a mensagem. Além da apresentação da mensagem a ser enviada, também será possível a visualização da mensagem criptografada a enviar e a ser recebida. Neste relatório também serão demonstrados o passo a passo da implementação do Chat, bem como as APIs que foram utilizadas, e também será apresentada uma breve conclusão.

2. Implementação

O trabalho abordado neste artigo foi desenvolvido em Java utilizando a IDE Eclipse, e foi utilizada a tecnologia de sockets para comunicação dos ambientes de comunicação (mensagens). Onde foi criado um modelo cliente-servidor e na troca de mensagem existe a criptografia da mensagem, do lado do cliente, passando pelo servidor, e sendo descriptografia chegando no cliente 2.

2.1 Diffie-Hellman

A criptografia abordada neste trabalho obedece a metodologia Diffie-Hellman, onde existe uma troca de chaves em um canal público. Onde existe uma troca de chaves segura para uma troca de mensagens criptografadas.

O modelo Diffie-Hellman não está limitado apenas a uma comunicação entre dois usuários, qualquer quantidade de usuários podem trocar mensagens simultaneamente, obedecendo o protocolo e trocando os dados intermediários entre si.

2.2 Criptografia

Para a segurança a criptografia é de extrema importância para a segurança da informação, a criptografia consiste na conversão de uma determinada informação em um formato que não seja legível a qualquer pessoa ou processo, ou seja, é o formato codificado de uma determinada informação. Os dados criptografados somente se tornam possíveis de leitura após os dados serem descriptografados, que o método inverso da criptografia, torna os dados ilegíveis em legíveis.

2.3 Socket

A tecnologia utilizada para comunicação entre os módulos de mensagens foi o Socket, uma tecnologia que nos permite realizar a comunicação distribuída em Java.

É estabelecida uma comunicação entre as máquinas e a troca de dados acontece pelo Socket. De acordo com a própria Oracle: “Socket é um ponto de comunicação entre duas máquinas”.

Na prática, o funcionamento baseia-se em criar a classe Servidora que tem a responsabilidade de esperar a conexão do cliente e a classe Cliente, que irá conectar-se ao Servidor.

Com isso temos um modelo cliente-servidor dentro da própria aplicação, onde podemos instanciar múltiplos clientes para comunicação a partir do socket de forma paralela.

2.4 Advanced Encryption Standard - AES

O Advanced Encryption Standard, conhecido pela sua sigla AES é um algoritmo de criptografia mais eficiente matematicamente e é indecifrável, o AES também é substituto do algoritmo de criptografia Data Encryption Standard – DES. O AES é considerado como padrão internacional de segurança de dados e foi efetivado como padrão do governo dos Estados Unidos no ano de 2002.

O AES é orientado a bytes e tem tamanhos diferentes de chaves iniciando em 128 bits até 256 bits, a criptografia utilizada é do tipo simétrica, ou seja, sua chave de criptografia e a chave de descriptografia são iguais. A estrutura que o algoritmo AES utilizada é a permutação-substituição, portanto realiza uma série de etapas de substituição e permutação para criptografar, o número de rodadas é definido pelo tamanho da chave:

- 128 bits: 10 rodadas;
- 192 bits: 12 rodadas;
- 256 bits: 14 rodadas;

2.5 Código fonte

Abaixo seguem *screenshots* de partes cruciais da implementação onde serão exibidas as questões de criptografia, comunicação entre os clientes e cálculos/tratamentos para permitir identificar os padrões de mensagens comunicadas.

```
System.out.println("[Cliente] Usuario " + cliente.getNmCliente() + ", recebendo a mensagem: " + txt + "\n");

//Verifica se a mensagem referente ao envio das chaves para o calculo da chave secreta compartilhada
if(isCrypto(txt)) {
    Long valorPublicoA = Long.valueOf(txt.split("\\|")[2].replaceAll("CRYPTO", ""));
    //Calculando valor publico B
    Long valorPublicoB = Encrypt.calculatePower(ServidorController.chavePublica2, cliente.getChavePrivada(), ServidorController.chavePublica1);
    System.out.println("Chave privada de "+ cliente.getNmCliente() + ": " + cliente.getChavePrivada());
    System.out.println("Chave publica de "+ cliente.getNmCliente() + ": " + ServidorController.chavePublica2);
    System.out.println("Valor publico de " + cliente.getNmCliente() + ": " + valorPublicoB);
    //Setando chave secreta compartilhada
    cliente.setChaveSecretaCompartilhada(Encrypt.calculatePower(valorPublicoA, cliente.getChavePrivada(), ServidorController.chavePublica1));
    System.out.println("Chave secreta compartilhada de " + cliente.getNmCliente() + ": " + cliente.getChaveSecretaCompartilhada() + "\n");
    //Envia a mensagem novamente, agora com a flag CRYPTOBACK
    String mensagem = cliente.getSocket().getLocalPort() + "|" + cliente.getNmCliente() + "|" + "CRYPTOBACK" + String.valueOf(valorPublicoB);
    enviar(mensagem);
//Verifica se a mensagem referente ao envio das chaves para o calculo da chave secreta compartilhada (RETORNO)
}else if(isCryptoBack(txt)) {
    Long valorPublicoB = Long.valueOf(txt.split("\\|")[2].replaceAll("CRYPTOBACK", ""));
    //Setando chave secreta compartilhada
    cliente.setChaveSecretaCompartilhada(Encrypt.calculatePower(valorPublicoB, cliente.getChavePrivada(), ServidorController.chavePublica1));
    System.out.println("Chave secreta compartilhada de " + cliente.getNmCliente() + ": " + cliente.getChaveSecretaCompartilhada());
    ServidorController.isCrypto = Boolean.TRUE;
}else {
    updateScreenFromServer(txt);
}
```

Acima, o segmento do código responsável por ouvir o cliente (parte do socket) e retornar a mensagem. A mensagem pode ser do tipo “crypto” que seria receber uma mensagem, e “cryptoBack” quando é uma resposta. Estes padrões foram estabelecidos para permitir o controle de mensagens.

```
public static String criarChaveDeAcordoComCrypto(Long chaveSecretaCompartilhada, TipoCryptoEnum tipoCrypto ) {
    String chaveSecretaCompartilhadaString = String.valueOf(chaveSecretaCompartilhada);

    for(int i = 0 ; i < 8 ; i++) {
        chaveSecretaCompartilhadaString = chaveSecretaCompartilhadaString.concat(chaveSecretaCompartilhadaString);
    }
    switch (tipoCrypto) {
   case C128: {
        return chaveSecretaCompartilhadaString.substring(0, TipoCryptoEnum.C128.getSize());
   }
   case C192: {
       return chaveSecretaCompartilhadaString.substring(0, TipoCryptoEnum.C192.getSize());
   }
   case C256:{
       return chaveSecretaCompartilhadaString.substring(0, TipoCryptoEnum.C256.getSize());
   }
   default:
   }
    return "";
}
```

Acima, código responsável por identificar o tamanho da chave de criptografia que será aplicada na mensagem. Essa criptografia acontece sempre no envio-recebimento, como também no inverso, logo, a mensagem ao trafegar sempre estará criptografada.

```
public static byte[] encrypt(String textopuro, String chaveencriptacao) throws Exception {
    System.out.println("Encriptando mensagem: " + textopuro);
    Cipher encripta = Cipher.getInstance("AES/CBC/PKCS5Padding", "SunJCE");
    SecretKeySpec key = new SecretKeySpec(chaveencriptacao.getBytes("UTF-8"), "AES");
    encripta.init(Cipher.ENCRYPT_MODE, key,new IvParameterSpec("AAAAAAAAAAAAAAAA".getBytes("UTF-8")));
    byte[] result =  encripta.doFinal(textopuro.getBytes("UTF-8"));
    System.out.println("Mensagem encriptada: " + new BigInteger(result) + "\n");
    return result;
}

public static String decrypt(byte[] textoencriptado, String chaveencriptacao) throws Exception{
    System.out.println("Decodificando mensagem: " + new BigInteger(textoencriptado));
    Cipher decripta = Cipher.getInstance("AES/CBC/PKCS5Padding", "SunJCE");
    SecretKeySpec key = new SecretKeySpec(chaveencriptacao.getBytes("UTF-8"), "AES");
    decripta.init(Cipher.DECRYPT_MODE, key,new IvParameterSpec("AAAAAAAAAAAAAAAA".getBytes("UTF-8")));
    String result =  new String(decripta.doFinal(textoencriptado),"UTF-8");
    System.out.println("Mensagem decodificada: " +  result);
    return result;
}
```

Acima temos a parte do código responsável por tratar e encriptar, como decriptar as mensagens trafegadas na rede, o mesmo recebe como parâmetro, a própria mensagem, como a chave gerada a partir do padrão Diffie-Hellman.

```java
public static boolean startServer(TipoCryptoEnum tipoCryptoEnum){
    try {
        tipoCrypto = tipoCryptoEnum;
        serverSocket = new ServerSocket(PORTA_PADRAO);
        imprimeInformacoesServidor();
        new Thread(new Runnable() {
            @Override
            public void run() {
                try{
                    while (true){
                        Socket s = serverSocket.accept();
                        socketsAceitos.add(s);
                        Thread.sleep(1000);
                        Cliente c = buscaCliente(s.getPort());
                        System.out.println("Usuario: " + c.getNmCliente() + " --> conectado!\n");
                        c.getEscutaCliente().exibirPainel();
                        new Thread(new EscutaServidor(s)).start();
                        Thread.sleep(200);
                    }
                } catch (Exception e){System.out.println(e.getMessage());}
            }
        }).start();
        return true;
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}
```

Local onde é realizada a inicialização do servidor (socket), e também dos clientes que estarão conectados para comunicação.

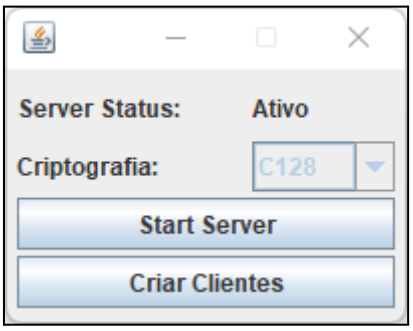
```java
private void enviaChaveSecretaCompartilhada() {
    System.out.println("Iniciando calculo de chave secreta compartilhada.\n");
    //Calcula valor publico
    Long valorPublico = Encrypt.calculatePower(ServidorController.chavePublica2, cliente.getChavePrivada(), ServidorController.chavePublica1);
    System.out.println("Chave privada de "+ cliente.getNmCliente() + ": " + cliente.getChavePrivada());
    System.out.println("Chave publica de "+ cliente.getNmCliente() + ": " + ServidorController.chavePublica1);
    System.out.println("Valor publico de " + cliente.getNmCliente() + ": " + valorPublico + "\n");
    //Monta a mensagem
    String mensagem = cliente.getSocket().getLocalPort() + "|" + cliente.getNmCliente() + "|" + "CRYPTO" + String.valueOf(valorPublico);
    //Envia a mensagem
    enviar(String.valueOf(mensagem));

}
```

```java
public static long calculatePower(long chavePublicaA, long chavePrivadaA, long chavePublicaB) {
    long result = 0;
    if (chavePrivadaA == 1){
        return chavePublicaA;
    }
    else{
        result = ((long)Math.pow(chavePublicaA, chavePrivadaA)) % chavePublicaB;
        return result;
    }
}
```

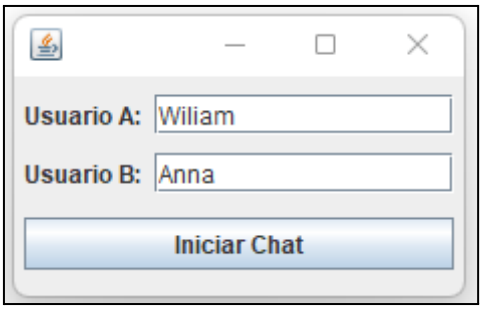
Por fim, o método enviaChaveSecretaCompartilhada chave a função calculatePower que será responsável por fazer o cálculo matemático, o Diffie Hellmann, para geração da chave secreta que será utilizada para estabelecer a conexão entre os dois clientes consumidores.

2.6 Interface

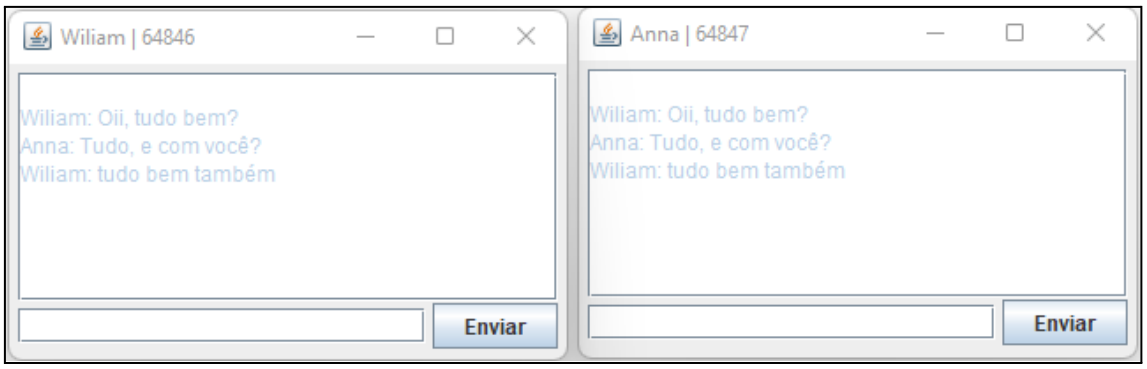
Abaixo serão listadas as telas da aplicação, desde a inicialização do servidor, até as telas responsáveis pela troca de mensagem.



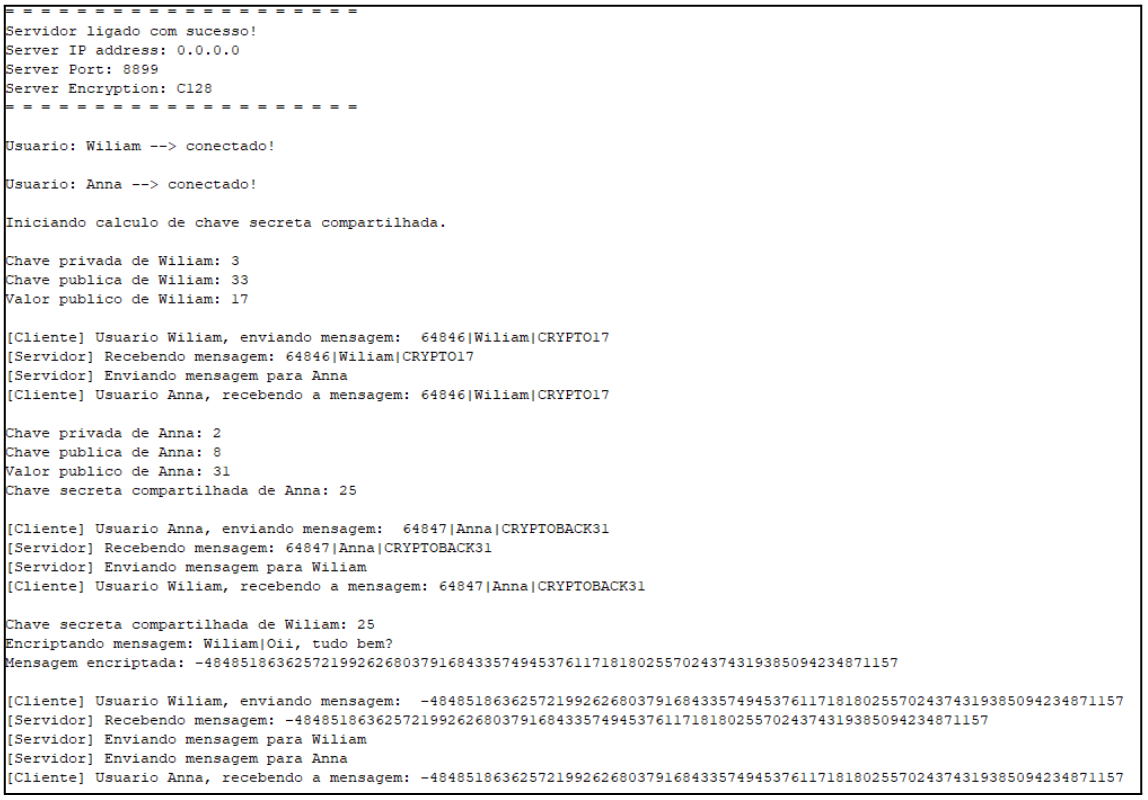
Tela responsável por iniciar o servidor, escolher o tamanho da chave de criptografia AES e inicialização dos clientes.



Após criar o servidor, e clicar em criar clientes, é aberta uma tela onde será possível declarar os clientes que irão realizar a comunicação.



Por fim, após iniciar o chat, são abertas as telas dos clientes, onde poderão ser realizadas as comunicações entre os clientes. Nesta troca de mensagens ocorre todos os cálculos e controle de chave (cliente e servidor) conforme retratado acima.



A partir da inicialização do servidor, todas as ações realizadas pelo software são exibidas no console, nele é possível acompanhar as trocas de chave, status, bem como as mensagens enviadas.

3. Conclusão

Com este trabalho foi possível observar tanto a execução de comunicação paralela entre múltiplos clientes, num modelo cliente-servidor, além da criptografia num modelo Diffie-Hallman a partir de um padrão AES para troca de mensagens seguras.

Com isso, observamos que existe um aumento na segurança, bem como na confidencialidade dos dados trocados nesse meio, onde a cada execução a chave é executada sobre a mensagem, realizando a sua criptografia, tornando-a ilegível sem a chave para decriptografia.

Referências

Kaspersky (2022). “O que é criptografia de dados? Definição e explicação”, https://www.kaspersky.com.br/resource-center/definitions/encryption, Novembro.

Devmedia (2015). “Java Socket: Entendendo a classe Socket e a ServerSocket em detalhes”, https://www.devmedia.com.br/java-socket-entendendo-a-classe-socket-e-a-serversocket-em-detalhes/31894, Novembro.

Kingston (2021). “O que é a criptografia de SSD e como funciona?”, https://www.kingston.com/br/blog/data-security/how-ssd-encryption-works, Novembro.

Kingston (2021). “O que é a criptografia de SSD e como funciona?”, https://www.kingston.com/br/blog/data-security/how-ssd-encryption-works, Novembro.

Abich, Geancarlo (2022). “Aula 10 – Criptografia”, disponível no ambiente virtual de aprendizagem da disciplina. Novembro.