UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

ENTREGA 6

Relatório Final

IF837 - Protocolos de Comunicação

Nicholas Henrique Justino Ferreira
Rafael Sutil Pereira

RECIFE, 23 DE AGOSTO DE 2021
Professor: Divanilson Rodrigo de Sousa Campelo

Sumário

1. Introdução

O presente trabalho tem dois grandes objetivos, o primeiro é a implementação de um protocolo de votação seguro em forma de biblioteca, e o segundo é desenvolver uma aplicação que utilize essa biblioteca. A aplicação deve atuar em uma rede Wi-Fi não confiável, de maneira a proporcionar confidencialidade, integridade e autenticidade para os usuários e o sistema. Visto que haverá a possibilidade de um invasor capturar, ler e injetar pacotes na rede, foram utilizadas técnicas de segurança para mitigar e impossibilitar ataques ao sistema. Tais técnicas foram aprendidas através dos conhecimentos adquiridos na disciplina de Protocolos de Comunicação (IF837) do Centro de Informática da Universidade Federal de Pernambuco.

2. Decisões Arquiteturais

Para atingir os objetivos deste projeto, adotamos o uso de ferramentas do ecossistema da Microsoft, como o .NET 5 utilizando a linguagem de programação C#, como mostrado na Entrega 1. Apesar deste projeto ter sido desenvolvido e testado no Windows 10, ele pode ser executado em outros sistemas operacionais, bastando ter o .NET 5 instalado. O modelo de comunicação utilizado foi o Cliente-Servidor, pois com ele conseguimos concentrar o gerenciamento da votação em um único servidor, onde diversos outros clientes poderão se conectar.

3. Problemas e Soluções

Ao longo desta seção vamos mostrar como foi desenvolvido o formato que o pacote tem para ser transmitido com segurança.

O primeiro grande problema que vamos solucionar é **Confidencialidade,** para isso vamos utilizar a chave pública do destinatário para criptografar a mensagem de forma que só o destinatário consiga decifrar. Mas temos que levar em consideração que todos possuem a chave pública, então o invasor poderia tentar decifrar as mensagens por tentativa e erro criptografando possíveis mensagens de voto para compará-las com as mensagens capturadas. Para resolvermos esse problema basta adicionar um número aleatório e único na mensagem impossibilitando que o invasor consiga adivinhar a mensagem.

A **Integridade** e a **Autenticidade** são garantidas da mesma forma, utilizando um código de autenticação de mensagem que usa uma chave criptográfica secreta para ser gerada, conhecida como HMAC. Para saber que a mensagem não foi alterada será feita uma comparação calculando o HMAC da mensagem que foi transmitida com o HMAC transmitido, processo pré-definido como Processo de Integridade. A autenticidade se dá pela chave secreta, essa chave é simétrica e é gerada uma nova a cada sessão iniciada, impossibilitando que o invasor faça um ataque de replay. Como o HMAC é único para cada sessão, é possível fazer um ataque de replay estando em uma mesma sessão? A resposta seria sim, mas adicionamos mais um campo no pacote que é o número de sequência, que será sempre verificado para saber se não há uma mensagem fora de ordem.

Outro problema que pode surgir é o *Truncation Attack*, que é quando o usuário recebe uma mensagem de encerramento falsa, com as metodologias adotadas acima este problema já teria sido resolvido, mas para adicionar mais uma camada de proteção, vamos colocar mais o campo **Tipo** no pacote, este campo só será adicionado nas mensagens enviadas do servidor para o cliente e terá valor 1 para mensagens comuns e 0 para mensagens de encerramento.

Os pacotes têm os formatos mostrados na Figura 1.

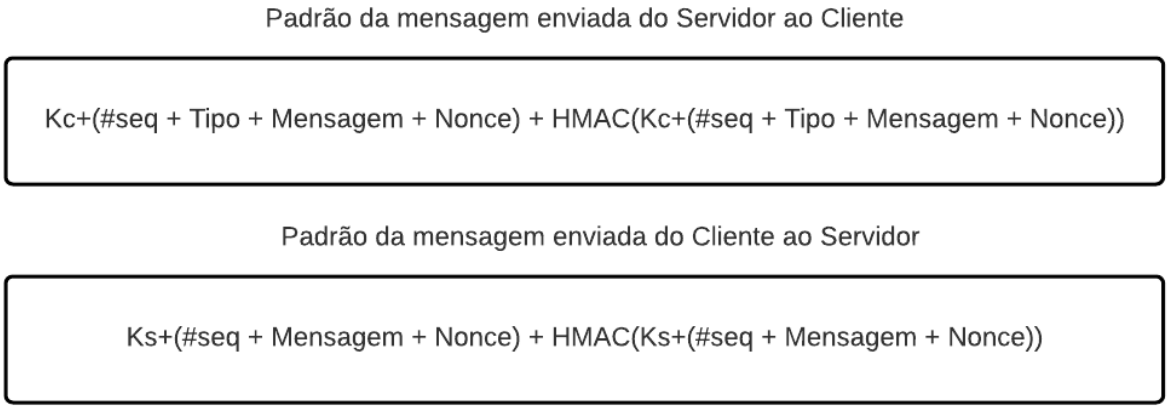

Figura 1: Padrão das mensagens

4. Handshake

Agora vamos descrever como será feita a troca das chaves públicas e a geração de uma chave secreta simétrica, processo pré-definido conhecido como Handshake, mostrado na Figura 2.

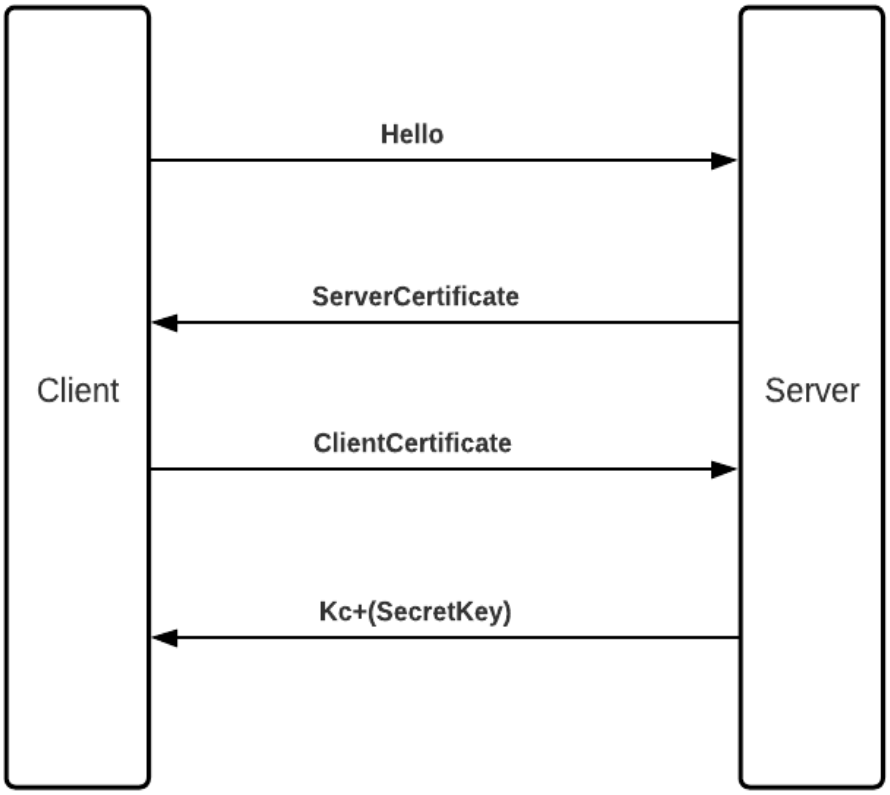

Figura 2: Processo de Handshake

i. O Cliente envia uma mensagem de Hello indicando que quer se conectar.
ii. O Servidor envia o seu certificado ao cliente.
iii. O Cliente verifica se o certificado é válido e autêntico e então envia o seu certificado.
iv. O Servidor cria uma chave secreta associada ao CPF do Cliente e envia usando a chave pública do Cliente.

Com esse processo foi possível o compartilhamento das chaves públicas por meio de certificados digitais, após isso o Cliente efetuará o login usando o CPF que está registrado no certificado.

5. Documentação do Protocolo

Para a Entrega 5, foi apresentada a documentação da implantação do protocolo. O protocolo VoteProtocol é constituído por duas classes, uma para ser usada com o servidor (PServer) e outra para ser usada com o cliente (PClient). Tais classes e suas funções são descritas a seguir:

5.1 Classe PServer

Representa uma instância do servidor com métodos e propriedades que auxiliam uma comunicação segura.

5.1.1 Handshake

Método estático e público que realiza a troca de certificados entre cliente e servidor, calcula e envia a chave secreta a fim de gerar o HMAC das mensagens. Este método não possui retorno.

Parâmetros

Socket current: Soquete do cliente no qual desejamos fazer a troca de mensagens.

5.1.2 PackMessage

Método estático e público que empacota uma mensagem adicionando o número de sequência, tipo e um número randômico, criptografando e concatenando com o HMAC da mensagem criptografada.

Parâmetros

string message: A mensagem que deve ser empacotada.
bool endMessage: Indica se o pacote é de encerramento de conexão.

Retorno

Retorna o pacote em um vetor de bytes pronto para ser transmitido com segurança.

Comentários

O tipo que será a mensagem é apenas para informar se é uma mensagem de encerramento ou não, ele foi adicionado para evitar possíveis *truncation attacks.*

5.1.3 UnPackMessage

Método estático e público que desempacota a mensagem recebida do cliente, verifica a integridade comparando o HMAC com o gerado localmente, verifica se o número de sequência é o esperado e o atualiza.

Parâmetros

byte[] package: Pacote a ser desempacotado.

Retornos

Retorna a mensagem descriptografada sem o número de sequência, sem o tipo e sem o HMAC. Caso o HMAC ou o número de sequência não coincida, o retorno será uma string "ERROR".

5.1.4 ReceiveResponseBytes

Método estático e privado que recebe um vetor de Bytes de um Socket conectado.

Parâmetros

Socket socket: Soquete do cliente que vamos receber os bytes

Retorno

Retorna os bytes recebidos.

Comentários

Este método, por ser privado, só será usado para receber o certificado do cliente no Handshake.

5.2 Classe PClient

Representa uma instância do cliente com métodos e propriedades que auxiliam uma comunicação segura entre cliente e servidor.

5.2.1 Handshake

Método público que inicia a comunicação com o servidor a fim de realizar a troca de certificados e receber a chave secreta para gerar o HMAC das mensagens. Não possui parâmetros e retorno.

Exceções

Exception Handshake Error: Esta exceção pode ser lançada por diversos erros que possam acontecer dentro do Handshake, dentre os possíveis erros temos a falha em tentar converter os bytes recebidos do servidor em um certificado, falha em extrair a chave pública deste certificado e falha na tentativa de descriptografar a mensagem que contém a SecretKey.

5.2.2 CertValidate

Método estático e privado que checa se o certificado recebido do servidor é válido. Não possui retorno, apenas lança exceções e desconecta o cliente caso haja erros.

Parâmetros

X509Certificate2 certificate: Certificado recebido do servidor.

Exceções

ArgumentNullException certificate: Esta exceção é lançada quando o certificado possui um valor nulo.
Exception Certificate was not issued by a trusted issuer: Esta exceção é lançada quando o certificado enviado não foi emitido pelo ElectionServer.

5.2.3 SendString

Método privado usado no Handshake para dar início a troca de mensagens. Não possui retorno.

Parâmetros

string text: Mensagem a ser codificada e enviada sem nenhuma proteção.

5.2.4 ReceiveResponseBytes

Método privado que é usado exclusivamente para receber o certificado do servidor. Não possui parâmetros.

Retornos

Retorna um vetor de 2048 bytes correspondente ao certificado do servidor.

5.2.5 ReceiveResponseKey

Método privado que é usado exclusivamente para receber a chave secreta do servidor no Handshake. Não possui parâmetros.

Retornos

Retorna um vetor de 288 bytes que representa a chave secreta.

5.2.6 ReceiveResponsePackage

Método público usado para receber o pacote que foi enviado pelo servidor. Não possui parâmetros.

Retornos

Retorna um vetor de 288 bytes que representa o pacote, 256 bytes para o mensagem criptografada e 32 bytes do HMAC.

5.2.7 PackMessage

Método público que empacota uma mensagem adicionando o número de sequência e um número randômico, criptografando e concatenando com o HMAC da mensagem criptografada.

Parâmetros

string message: A mensagem que deve ser empacotada.

Retornos

Retorna o pacote em um vetor de bytes pronto para ser transmitido com segurança.

5.2.8 UnPackMessage

Método público que desempacota a mensagem recebida do servidor, verifica a integridade comparando o HMAC com o gerado localmente, verifica se o número de sequência é o esperado e o atualiza.

Parâmetros

byte[] package: Pacote a ser desempacotado

Retornos

Retorna a mensagem descriptografada sem o número de sequência, sem o tipo e sem o HMAC.

Exceções

Caso o HMAC não coincida, o retorno será uma string "Message without integrity". Caso o número de sequência não coincida, o retorno será uma string "Sequence number does not match".

5.2.9 CalculateSHA256

Método público que calcula o valor da função SHA256 sobre um string.

Parâmetros

String value: string a ser codificada em SHA256.

Retornos

Sb.ToString(): string do valor SHA256 calculado

6. Aplicação

Agora apresenta-se o funcionamento de uma aplicação que implementa este protocolo. Como dito nas decisões arquiteturais, usou-se o modelo Cliente-Servidor. Vale ressaltar que toda a aplicação foi desenvolvida na linguagem de programação C#.

Ao iniciar o Servidor será solicitado o nome da votação, a data que a votação deve encerrar, o máximo de votos que a votação deverá ter, e se deseja usar os candidatos pré-definidos ou se deseja adicionar os candidatos manualmente. Caso a segunda opção seja escolhida os candidatos devem ser adicionados no formato “ID,Nome”, e para finalizar deverá ser digitado a palavra “exit”. Com isso, o Servidor mostrará em tempo real o resultado da eleição após o primeiro voto realizado.

Como o Servidor possui um número de sequência para as mensagens e também uma chave secreta armazenada, não podemos atender dois clientes ao mesmo tempo, por isso foi criada uma fila de espera. Cada conexão de Cliente tem o tempo limite de 1 minuto para realizar a votação, caso esse tempo seja atingido sem o Cliente finalizar, a conexão será encerrada e nenhum voto será contabilizado. Este tempo limite serve para não causar um congestionamento na fila de clientes que querem se conectar.

Quando o Cliente é iniciado, há uma verificação se o servidor está ocupado, se estiver ocupado, o Cliente vai para o final da fila, caso contrário será iniciado o timer, e as credenciais serão solicitadas. Vale ressaltar que o CPF usado no login deverá ser o mesmo CPF usado no handshake ao enviar o certificado. Outra coisa que é verificada após o Login é se o Cliente já votou, caso ele tenha votado e a eleição ainda esteja ativa o Cliente também será desconectado. Caso o Login seja bem-sucedido, o Servidor enviará a lista de candidatos disponíveis para voto, e basta o Cliente digitar o ID do candidato, após isso uma mensagem de agradecimento é enviada, o voto é contabilizado e o Cliente é desconectado, abrindo vaga para outros clientes votarem.

Após passar o tempo limite da votação ou o número máximo de votos, qualquer Cliente poderá voltar a fazer Login para verificar o resultado da eleição.

A Figura 3, a seguir, mostra o fluxo de mensagens entre Cliente e Servidor.

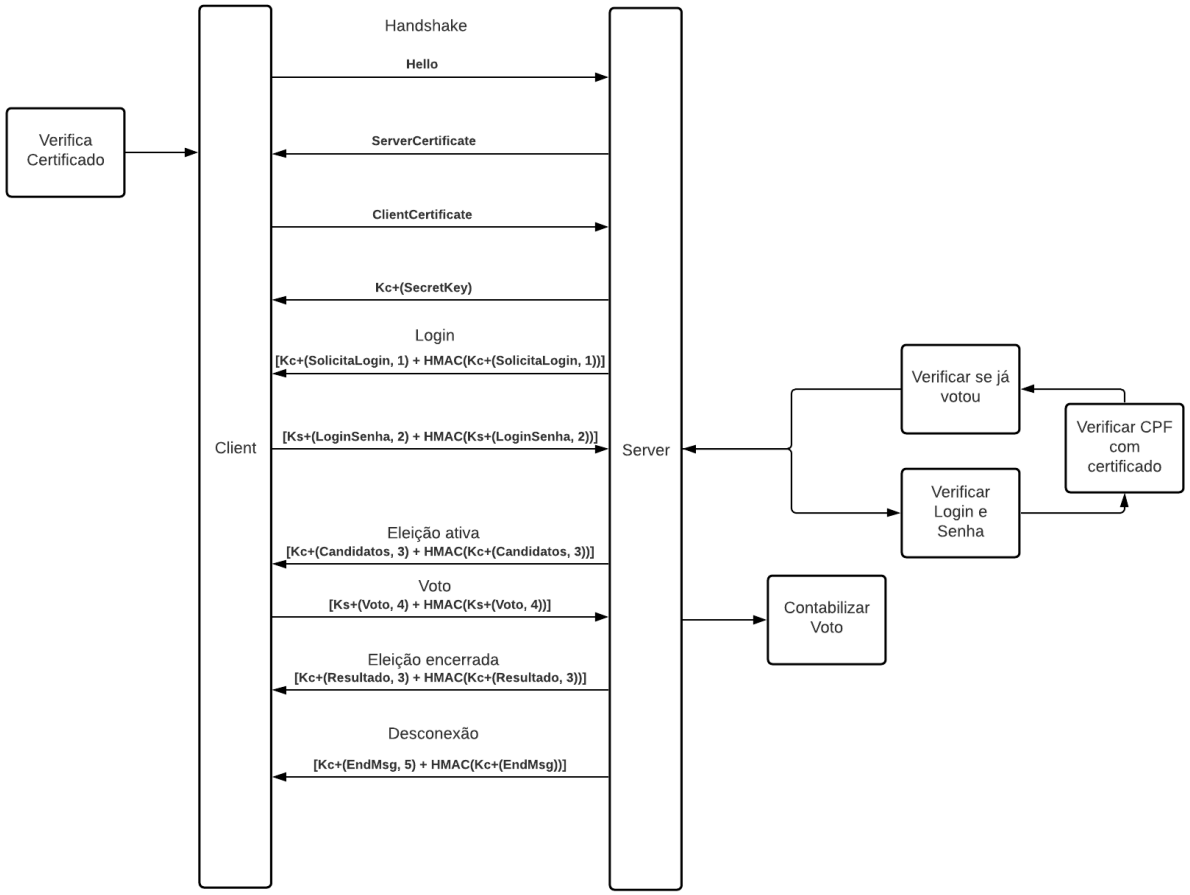

Figura 3: Fluxo de mensagens entre Cliente e Servidor

7. Conclusão

Este projeto teve como objetivo a implantação de um sistema de votação seguro, VoteSystem, em uma rede Wi-Fi não confiável, para tal feito foram realizadas entregas parciais do projeto com propósito de exemplificar cada etapa do projeto. Neste relatório final, foram apresentadas a metodologia (Decisões Arquiteturais), processos de segurança, fluxo de mensagens, design do protocolo junto com a documentação do protocolo e sua aplicação. As técnicas de segurança utilizadas foram definidas com base na possibilidade de um invasor, que pode capturar, ler e injetar pacotes na rede, não poder interferir na votação. Ou seja, este trabalho visa o estabelecimento da confidencialidade, autenticidade e integridade das mensagens trocadas entre Cliente e Servidor para que não haja fraude e quebra do sigilo do voto.