Atividade de Criptografia | EC38D | Segurança e Auditoria de Sistemas | Professor Dr. Lucas Sampaio
Aluno Guilherme Henrique G Oliveira 1913298

Introdução

Este trabalho tem como objetivo criar um programa de chat via socket de tal forma que não seja possível observar o comportamento do sniffer da rede. Como referência, utilizou-se o material fornecido pelo professor através do Moodle da disciplina. Como houve a necessidade de criar várias classes de abstrações para refatoramento do código, foi escolhido a linguagem de programação Typescript com NodeJS 16.0.0 como motor de execução.

O chat recriado manteve interface parecida com o programa original fornecido pelo professor. Um servidor socket é ligado podendo escolher sua interface de rede e porta de operação. E *n* clientes sockets são conectados a este servidor também podendo escolher o endereço IP e porta do servidor que deseja se conectar.

Para arquitetura cliente-servidor, foi desenvolvido um protocolo simples de comunicação inspirado em *AT Command*, porém com valores formatados em *JSON*. Detalhe sobre este protocolo está no primeiro capítulo.

Para garantir confidencialidade, autenticidade e integridade da comunicação foram desenvolvidos três fluxos discorridos no capítulo 2.

Por fim verificou-se que a política de segurança é satisfeita por meio de um sniffer de rede, no caso, foi utilizado o software *WireShark*.

O código fonte do trabalho está disponível no repositório online no endereço https://github.com/Guihgo/lab-chat-p2p. Todas informações de como executar o código se encontram no README.md na raiz do repositório.

1. Protocolo de comunicação

<OP_CODE>=<DATA>[,HMAC(DATA)]

OP_CODE: String de operação. Podendo ser: AUTH | ERROR | HANDSHAKE | SEND

DATA: String no formato JSON contendo tipos predefinidos para cada tipo de operação

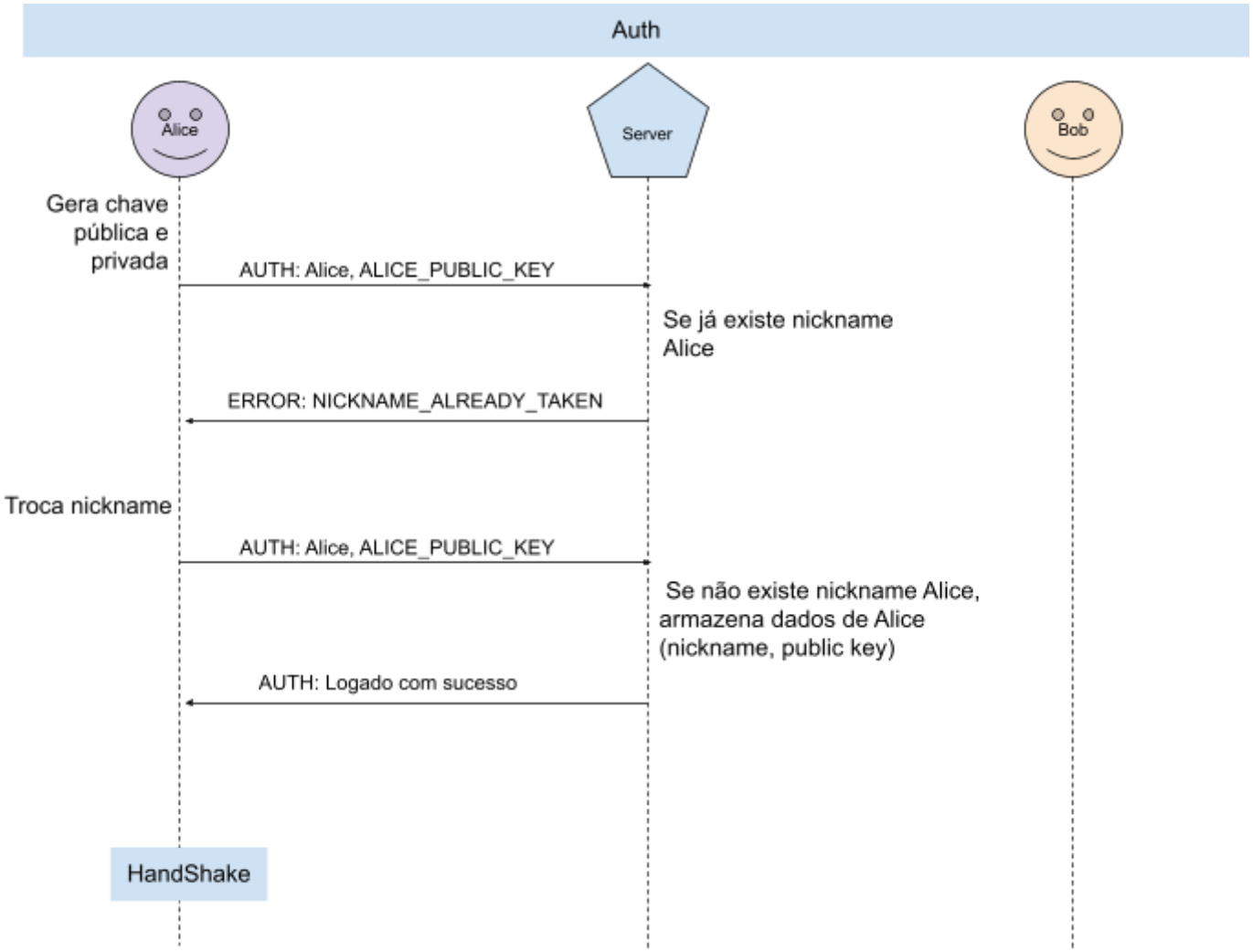
HMAC (hash-based message authentication code): hash SHA256 feito em cima da DATA enviada para o conceito de integridade

Exemplo de payload (op + data):

```
AUTH={“nickname”:”Alice”, “publicKey”:”-----BEGIN PUBLIC KEY----- MIIBIjA…”}
```

2. Confidencialidade, autenticidade e integridade da comunicação

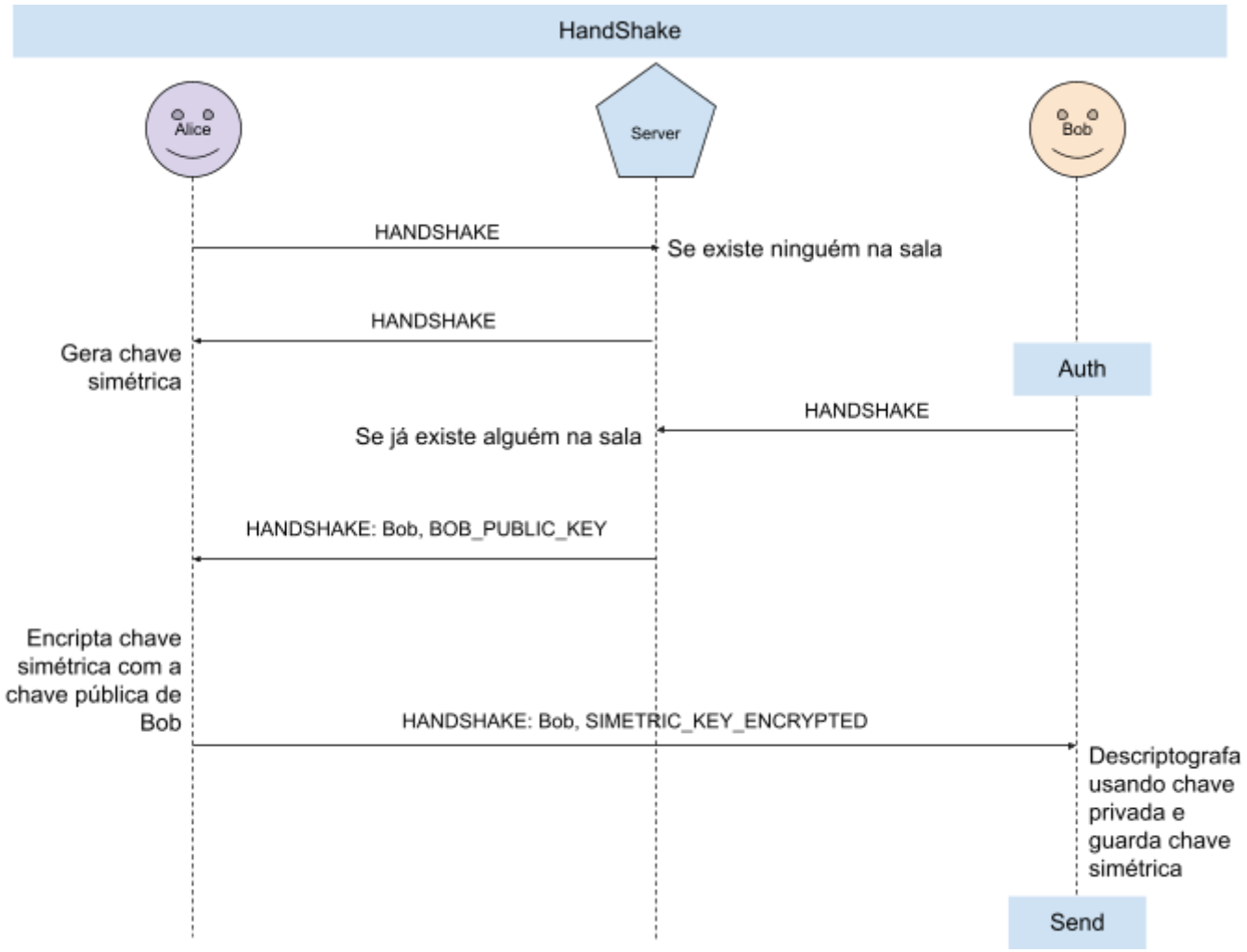
O fluxo “Auth” tem objetivo de autenticar o cliente no servidor. O método de autenticação é por nickname único, sem senha. Desta forma, apenas um único cliente poderia se autenticar por vez utilizando um único nickname.



Fluxo Auth

Atividade de Criptografia | EC38D | Segurança e Auditoria de Sistemas | Professor Dr. Lucas Sampaio
Aluno Guilherme Henrique G Oliveira 1913298

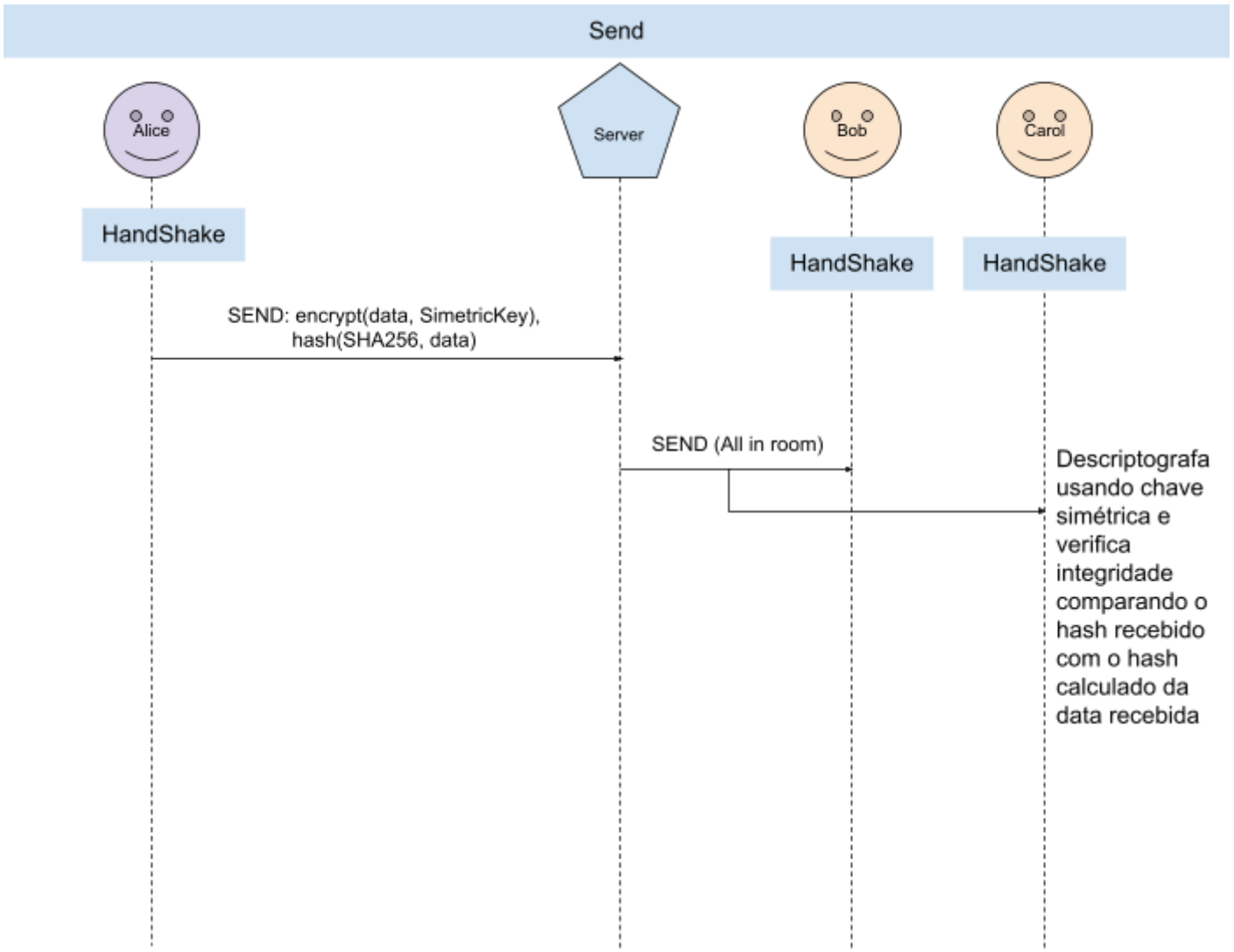
O fluxo “HandShake” tem objetivo de gerenciar a chave simétrica utilizada para criptografar as mensagens enviadas de um cliente para outro. Uma chave simétrica é gerada em um dos primeiros clientes que se conecta ao servidor. Os clientes seguintes requisitam a chave simétrica do primeiro cliente para então estarem habilitados a enviar mensagens criptografadas. Para transportar a chave simétrica de forma segura a um sniffer de rede, o cliente solicitador, portador de um par de chave assimétrica (pública e privada), fornece sua chave pública para o cliente solicitado e portador da chave simétrica para que este último possa criptografar sua chave simétrica utilizando a chave pública do cliente solicitador. Desta forma, somente o cliente solicitador consegue descriptografar a chave simétrica criptografada com sua chave pública utilizando sua chave privada.



Fluxo HandShake

Atividade de Criptografia | EC38D | Segurança e Auditoria de Sistemas | Professor Dr. Lucas Sampaio
Aluno Guilherme Henrique G Oliveira 1913298

O fluxo “Send” tem objetivo de enviar mensagens, de forma segura, para todos clientes conectados no servidor, uma vez que foi finalizado o fluxo de HandShake e sincronizado a chave simétrica entre os clientes. Basicamente, utiliza-se de criptografia simétrica para proteger a mensagem trafegada na rede e o cliente receptor da mensagem, com a mesma chave simétrica, descriptografa a mensagem e exibe para o usuário.



Fluxo Send

Atividade de Criptografia | EC38D | Segurança e Auditoria de Sistemas | Professor Dr. Lucas Sampaio
Aluno Guilherme Henrique G Oliveira 1913298

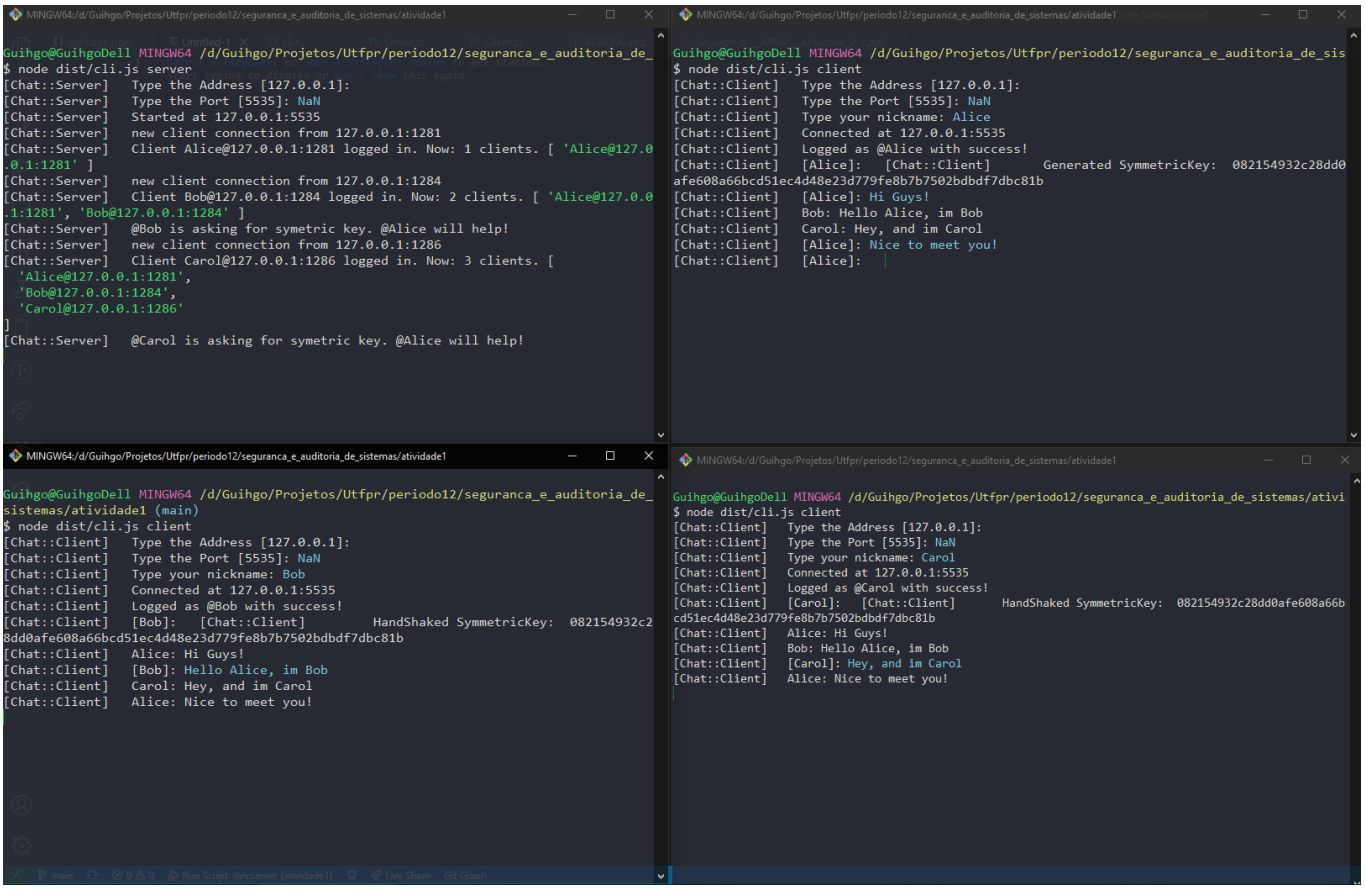
3. Resultados

Performance

O chat foi testado com 3 conexões simultâneas utilizado em média 13,5 Mb da memória RAM e um valor insignificante da CPU.

0%	13,7 MB	"C:/Program Files/nodejs/node.exe" dist/cli.js client
0%	13,6 MB	"C:/Program Files/nodejs/node.exe" dist/cli.js client
0%	13,2 MB	"C:/Program Files/nodejs/node.exe" dist/cli.js client
0%	13,1 MB	"C:/Program Files/nodejs/node.exe" dist/cli.js server

Uso da memória



Chat em 3 conexões simultâneas e servidor

Atividade de Criptografia | EC38D | Segurança e Auditoria de Sistemas | Professor Dr. Lucas Sampaio
Aluno Guilherme Henrique G Oliveira 1913298

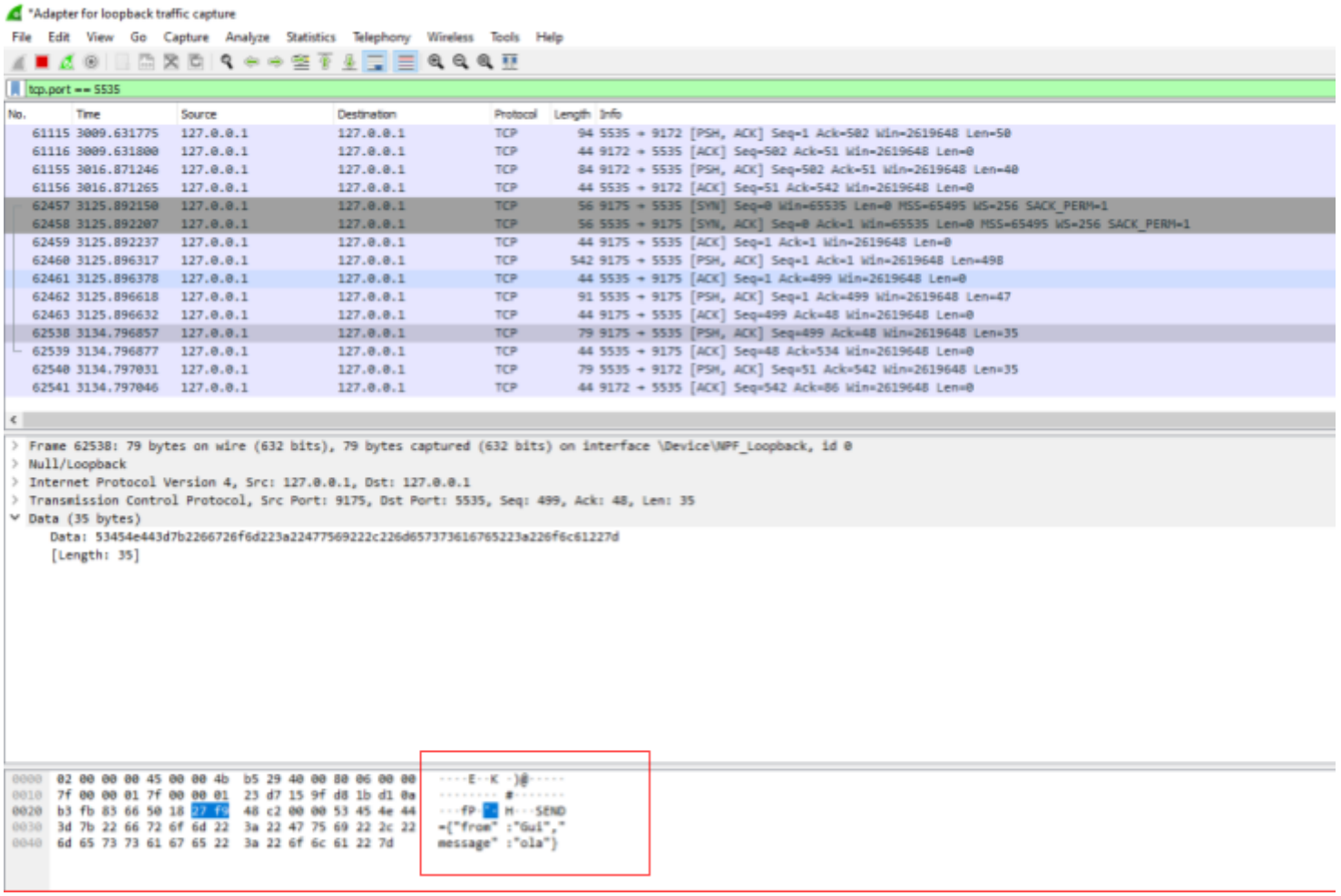
Segurança

No seguinte cenário, o *client* cujo nickname “Gui” envia a mensagem “ola” para a sala.

Chat sem implementação do fluxo HandShake e Send

Antes da implementação de HandShake entre os *clients* foi possível visualizar pelo *sniffer* o conteúdo dos pacotes transmitidos por tcp

Atividade de Criptografia | EC38D | Segurança e Auditoria de Sistemas | Professor Dr. Lucas Sampaio
Aluno Guilherme Henrique G Oliveira 1913298

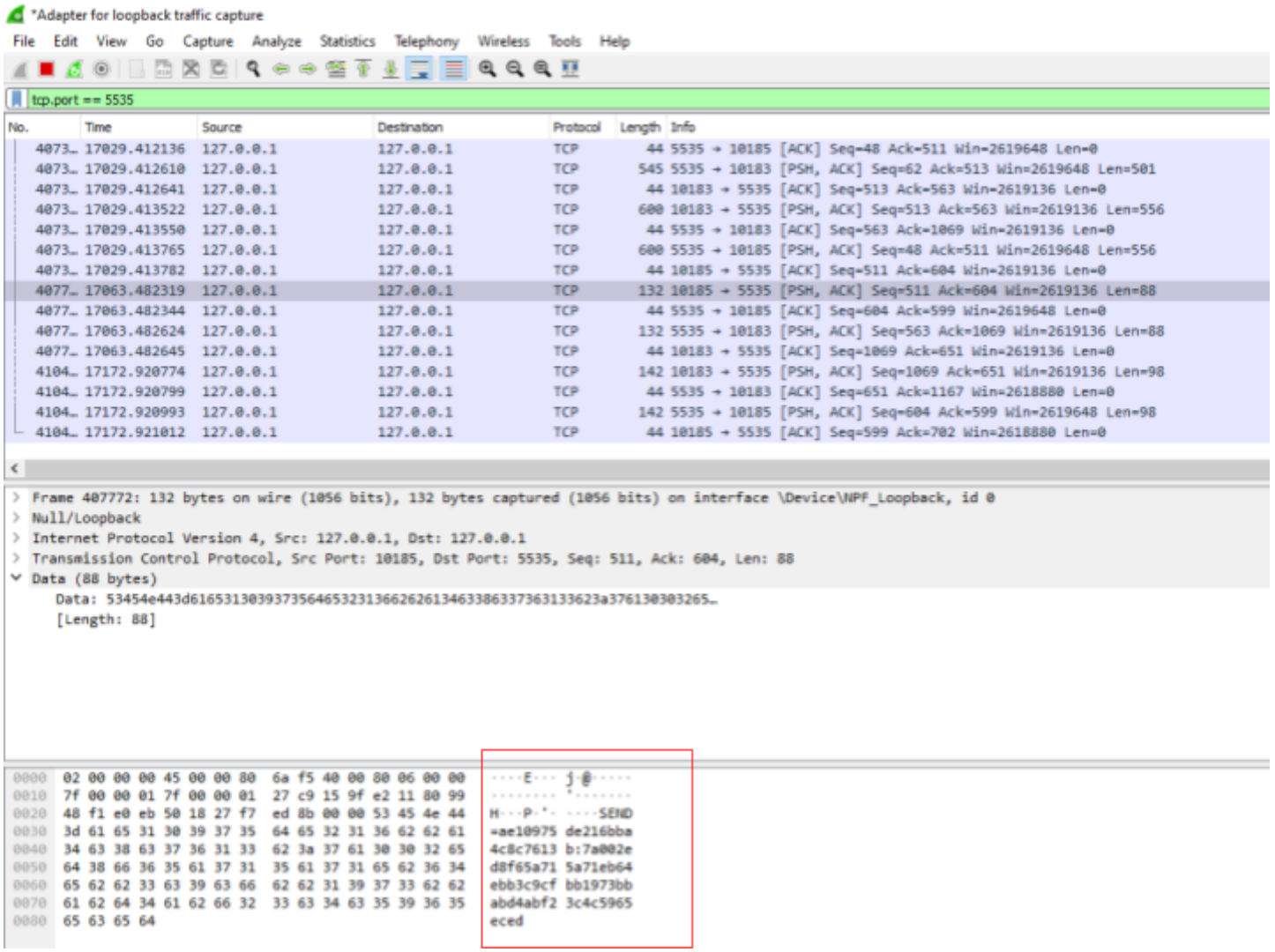


Sniff sobre chat sem implementação do fluxo HandShake e Send

Após a implementação do HandShake por Criptografia assimétrica e simétrica não foi mais possível visualizar dados sensíveis *snifados*. Pois agora os dados trafegados estavam criptografados por chave simétrica negociada por chave assimétrica.

Chat com implementação do fluxo HandShake e Send

Atividade de Criptografia | EC38D | Segurança e Auditoria de Sistemas | Professor Dr. Lucas Sampaio
Aluno Guilherme Henrique G Oliveira 1913298



Sniff sobre chat com implementação do fluxo HandShake e Send

4. Conclusão

Apesar da inclusão de mecanismos de segurança implementados no decorrer do trabalho, há outras oportunidades de se aprimorar a segurança diminuindo vulnerabilidades. Por exemplo, criar autenticação por senha, bloquear o acesso ao servidor por arranjo de IPs não permitidos (firewall, whitelist, …). Desta forma um cliente malicioso não consegue conectar-se ao servidor somente para obter a chave simétrica. Outra questão que este trabalho não contemplou é que a criptografia ponta a ponta não se estendeu pelo payload completo (OP_CODE + DATA) e somente se utilizou na DATA enviada. Desta forma um sniff consegue monitorar as atividades da rede, por exemplo, se um usuário está enviando informação ou se conectando. Uma solução para isso é implementar o TLS na conexão socket, desta forma, encapsulando todo dado trafegado.

Cabe ressaltar que o objetivo do trabalho foi alcançado com êxito visto os objetivos definidos na atividade.

Observação final

O professor aconselhou fazer a assinatura da mensagem enviada utilizando a chave privada da origem e verificar, quando receber a mensagem, utilizando a chave pública da origem a fim de atingir o conceito de integridade. Neste trabalho, utilizou-se HMAC utilizando a chave simétrica em cima da mensagem para atingir tal conceito, pois fazer o hash SHA256 tem menor custo computacional do que fazer a assinatura com uma chave privada de 2048 *bytes* e depois verificar a assinatura com a chave pública. E como este processo é realizado para toda mensagem enviada e recebida, o custo computacional é relevante, ainda mais quando se trata de dispositivos móveis cujos processadores são inferiores.

5. Referências

Crypto | Node.js v16.17.0 Documentation. **Node JS ORG**, 2022. Disponível em: https://nodejs.org/dist/latest-v16.x/docs/api/crypto.html . Acesso em: 07 de setembro de 2022.

Computational and Energy Costs of Cryptographic Algorithms on Handheld Devices. **Rifà-Pous, Helena and Herrera-Joancomartí, Jordi**, 2011. Disponível em: https://www.mdpi.com/1999-5903/3/1/31 . Acesso em: 07 de setembro de 2022.