



## 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 percorridos 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

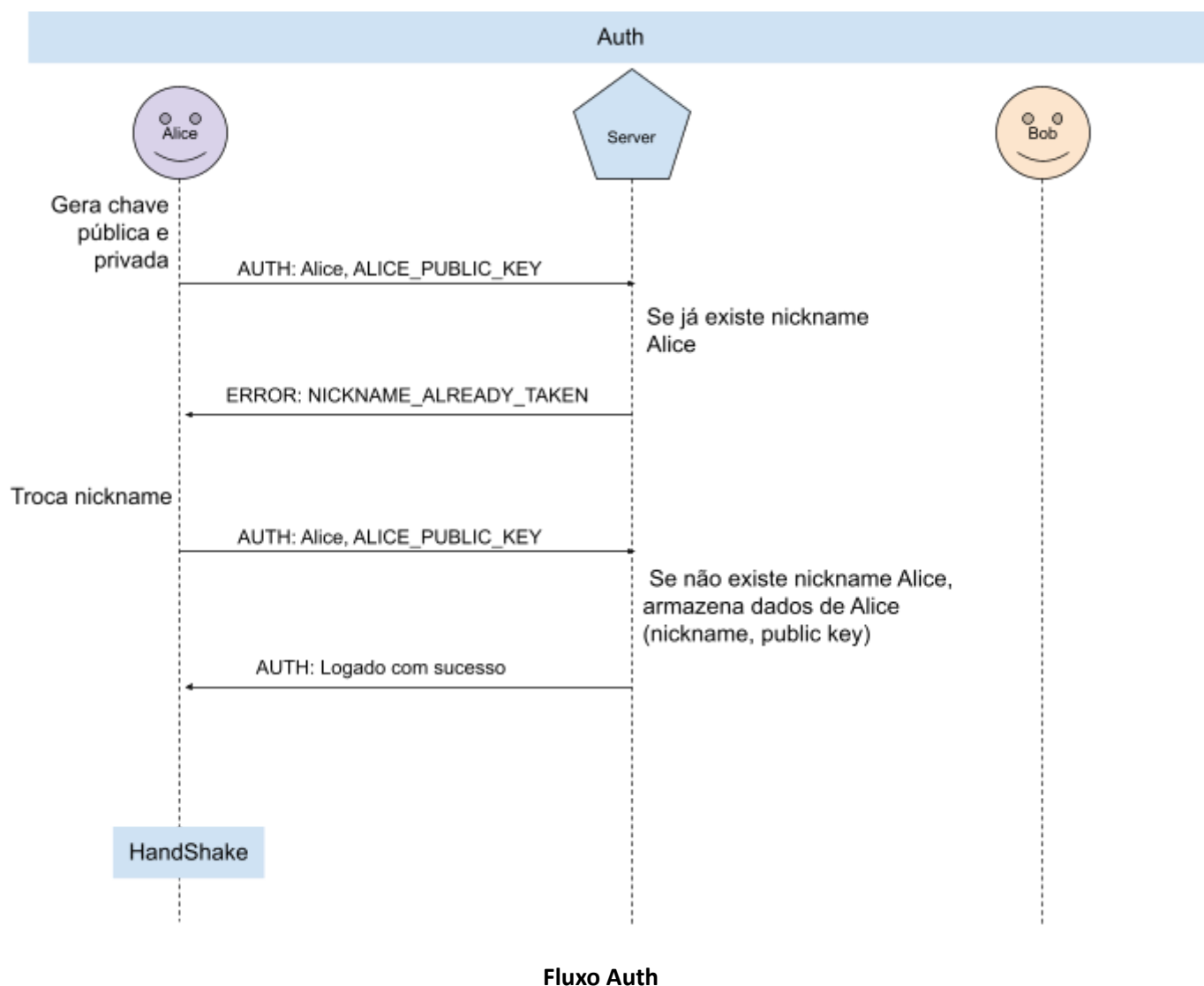
Exemplo de payload (op + data):

```
AUTH={"nickname":"Alice", "publicKey":"-----BEGIN PUBLIC KEY----- MIIBIjA..."}
```



## 2. Confidencialidade, autenticidade e integridade da comunicação

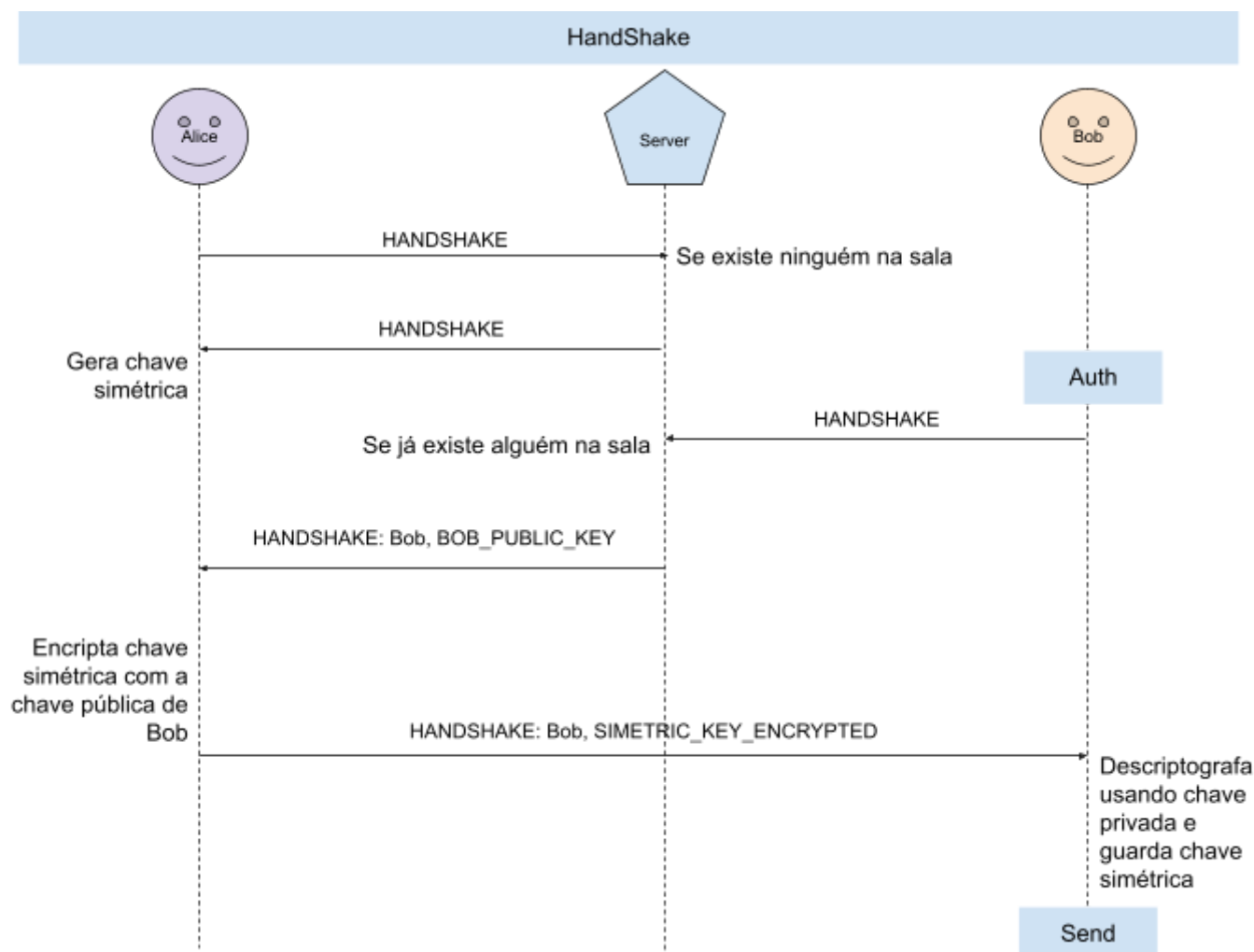
O fluxo "Auth" tem objetivo de autenticar o cliente no servidor. O método de autenticação é por nickname único, sem senha. Desta forma, apenas um único cliente poderia se autenticar por vez utilizando um único nickname.





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.

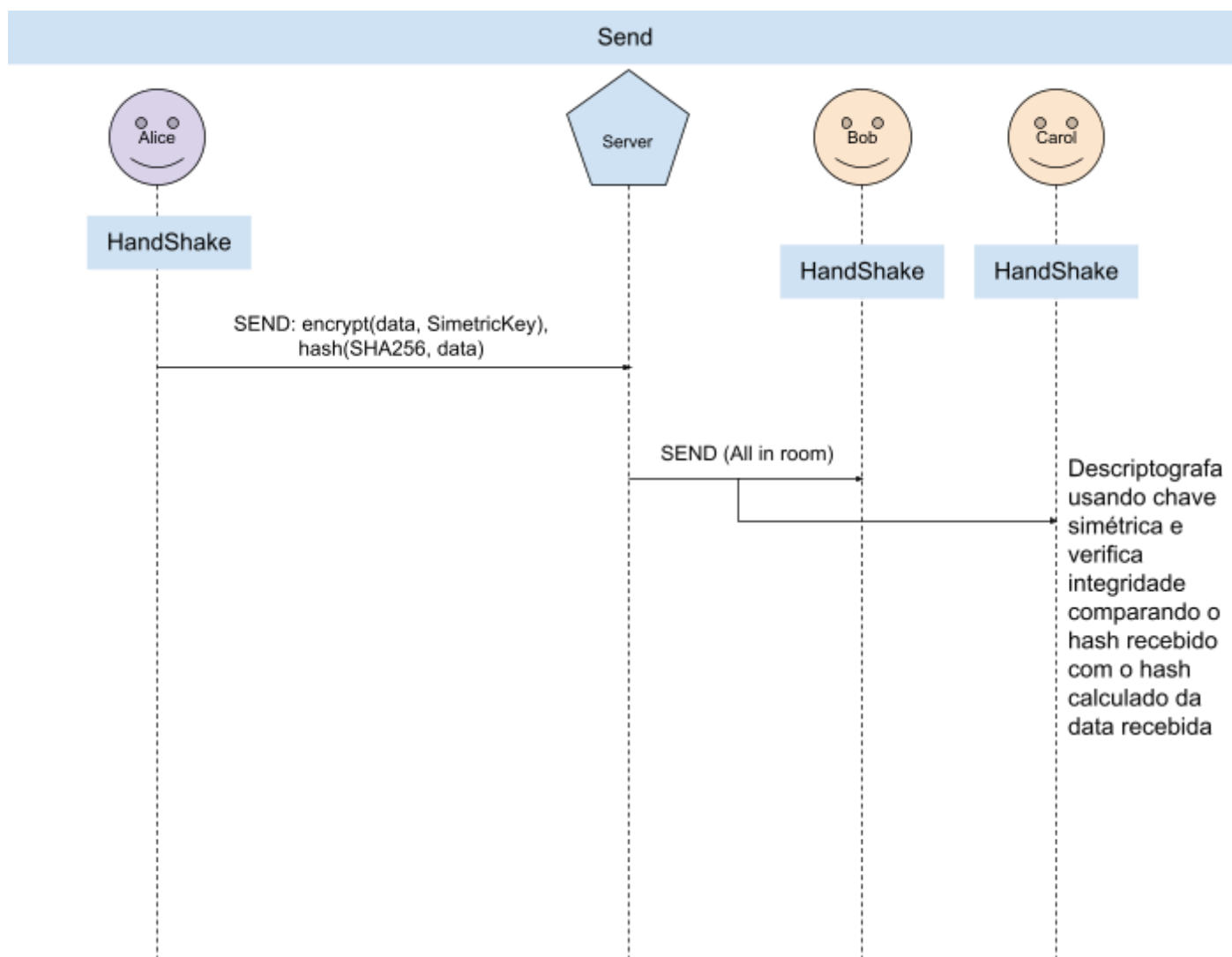


Fluxo HandShake



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

```
MINGW64/d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
Guihgo@GuihgoDell MINGW64 /d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
$ node dist/cli.js server
[Chat::Server] Type the Address [127.0.0.1]:
[Chat::Server] Type the Port [5535]: NaN
[Chat::Server] Started at 127.0.0.1:5535
[Chat::Server] new client connection from 127.0.0.1:1281
[Chat::Server] Client Alice@127.0.0.1:1281 logged in. Now: 1 clients. [ 'Alice@127.0.0.1:1281' ]
[Chat::Server] new client connection from 127.0.0.1:1284
[Chat::Server] Client Bob@127.0.0.1:1284 logged in. Now: 2 clients. [ 'Alice@127.0.0.1:1281', 'Bob@127.0.0.1:1284' ]
[Chat::Server] @Bob is asking for symmetric key. @Alice will help!
[Chat::Server] new client connection from 127.0.0.1:1286
[Chat::Server] Client Carol@127.0.0.1:1286 logged in. Now: 3 clients. [ 'Alice@127.0.0.1:1281', 'Bob@127.0.0.1:1284', 'Carol@127.0.0.1:1286' ]
[Chat::Server] @Carol is asking for symmetric key. @Alice will help!

MINGW64/d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
Guihgo@GuihgoDell MINGW64 /d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
$ node dist/cli.js client
[Chat::Client] Type the Address [127.0.0.1]:
[Chat::Client] Type the Port [5535]: NaN
[Chat::Client] Type your nickname: Alice
[Chat::Client] Connected at 127.0.0.1:5535
[Chat::Client] Logged as @Alice with success!
[Chat::Client] [Alice]: [Chat::Client] Generated SymmetricKey: 082154932c28dd0afe608a66bcd51ec4d48e23d779fe8b7b7502bdbdf7dbc81b
[Chat::Client] [Alice]: Hi Guys!
[Chat::Client] Bob: Hello Alice, im Bob
[Chat::Client] Carol: Hey, and im Carol
[Chat::Client] [Alice]: Nice to meet you!
[Chat::Client] [Alice]:

MINGW64/d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
Guihgo@GuihgoDell MINGW64 /d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
$ node dist/cli.js client
[Chat::Client] Type the Address [127.0.0.1]:
[Chat::Client] Type the Port [5535]: NaN
[Chat::Client] Type your nickname: Bob
[Chat::Client] Connected at 127.0.0.1:5535
[Chat::Client] Logged as @Bob with success!
[Chat::Client] [Bob]: [Chat::Client] HandShaked SymmetricKey: 082154932c28dd0afe608a66bcd51ec4d48e23d779fe8b7b7502bdbdf7dbc81b
[Chat::Client] Alice: Hi Guys!
[Chat::Client] [Bob]: Hello Alice, im Bob
[Chat::Client] Carol: Hey, and im Carol
[Chat::Client] Alice: Nice to meet you!

MINGW64/d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
Guihgo@GuihgoDell MINGW64 /d/Guihgo/Projetos/Utfpr/periodo12/seguranca_e_auditoria_de_sistemas/atividade1
$ node dist/cli.js client
[Chat::Client] Type the Address [127.0.0.1]:
[Chat::Client] Type the Port [5535]: NaN
[Chat::Client] Type your nickname: Carol
[Chat::Client] Connected at 127.0.0.1:5535
[Chat::Client] Logged as @Carol with success!
[Chat::Client] [Carol]: [Chat::Client] HandShaked SymmetricKey: 082154932c28dd0afe608a66bcd51ec4d48e23d779fe8b7b7502bdbdf7dbc81b
[Chat::Client] Alice: Hi Guys!
[Chat::Client] Bob: Hello Alice, im Bob
[Chat::Client] [Carol]: Hey, and im Carol
[Chat::Client] Alice: Nice to meet you!
```

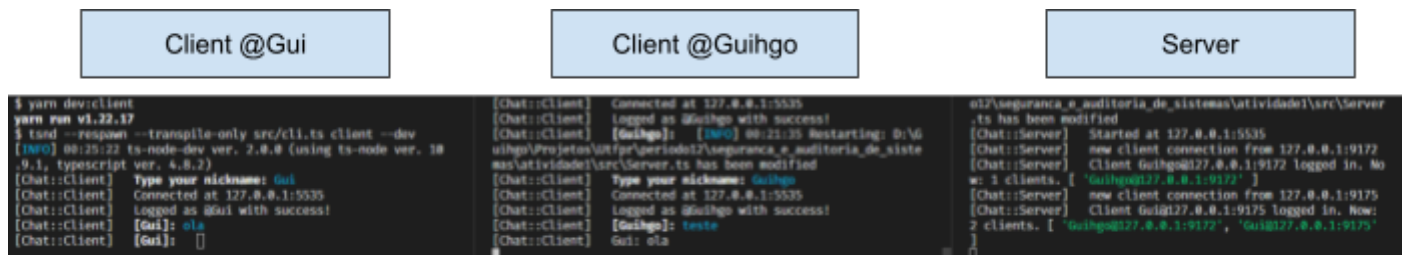
Chat em 3 conexões simultâneas e servidor



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

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 5535

No.	Time	Source	Destination	Protocol	Length	Info
61115	3009.631775	127.0.0.1	127.0.0.1	TCP	94	5535 → 9172 [PSH, ACK] Seq=1 Ack=502 Win=2619648 Len=50
61116	3009.631800	127.0.0.1	127.0.0.1	TCP	44	9172 → 5535 [ACK] Seq=502 Ack=51 Win=2619648 Len=0
61155	3016.871246	127.0.0.1	127.0.0.1	TCP	84	9172 → 5535 [PSH, ACK] Seq=502 Ack=51 Win=2619648 Len=40
61156	3016.871265	127.0.0.1	127.0.0.1	TCP	44	5535 → 9172 [ACK] Seq=51 Ack=542 Win=2619648 Len=0
62457	3125.892150	127.0.0.1	127.0.0.1	TCP	56	9175 → 5535 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
62458	3125.892207	127.0.0.1	127.0.0.1	TCP	56	5535 → 9175 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
62459	3125.892237	127.0.0.1	127.0.0.1	TCP	44	9175 → 5535 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
62460	3125.896317	127.0.0.1	127.0.0.1	TCP	542	9175 → 5535 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=498
62461	3125.896378	127.0.0.1	127.0.0.1	TCP	44	5535 → 9175 [ACK] Seq=1 Ack=499 Win=2619648 Len=0
62462	3125.896618	127.0.0.1	127.0.0.1	TCP	91	5535 → 9175 [PSH, ACK] Seq=1 Ack=499 Win=2619648 Len=47
62463	3125.896632	127.0.0.1	127.0.0.1	TCP	44	9175 → 5535 [ACK] Seq=499 Ack=48 Win=2619648 Len=0
62538	3134.796857	127.0.0.1	127.0.0.1	TCP	79	9175 → 5535 [PSH, ACK] Seq=499 Ack=48 Win=2619648 Len=35
62539	3134.796877	127.0.0.1	127.0.0.1	TCP	44	5535 → 9175 [ACK] Seq=48 Ack=534 Win=2619648 Len=0
62540	3134.797031	127.0.0.1	127.0.0.1	TCP	79	5535 → 9172 [PSH, ACK] Seq=51 Ack=542 Win=2619648 Len=35
62541	3134.797046	127.0.0.1	127.0.0.1	TCP	44	9172 → 5535 [ACK] Seq=542 Ack=86 Win=2619648 Len=0

<

> Frame 62538: 79 bytes on wire (632 bits), 79 bytes captured (632 bits) on interface \Device\NPF\_{Loopback, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 9175, Dst Port: 5535, Seq: 499, Ack: 48, Len: 35

> Data (35 bytes)

Data: 53454e443d7b2266726f6d223a22477569222c226d657373616765223a226f6c61227d

[Length: 35]

0000 02 00 00 00 45 00 00 4b b5 29 40 00 00 06 00 00 ---E-K-)----

0010 7f 00 00 01 7f 00 00 01 23 d7 15 9f d8 1b d1 0e ---FP H---SEND

0020 b3 fb 83 66 50 18 27 6e 48 c2 00 00 53 45 4e 44 ---[{"from": "Gui",

0030 3d 7b 22 66 72 6f 6d 22 3a 22 47 75 69 22 2c 22 =["from": "Gui",

0040 6d 65 73 73 61 67 65 22 3a 22 6f 6c 61 22 7d message": "ola"]

### 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.

Server	Client @Alice	Client @Bob
<pre>\$ node dist/c1.js server [Chat::Server] Type the Address [127.0.0.1]: [Chat::Server] Type the Port [5535]: 5535 [Chat::Server] Started at 127.0.0.1:5535 [Chat::Server] new client connection from 127.0.0.1:18183 [Chat::Server] Client Alice(127.0.0.1:18183) logged in. Now: 1 clients, [ Alice(127.0.0.1:18183) ] [Chat::Server] new client connection from 127.0.0.1:18185 [Chat::Server] Client Bob(127.0.0.1:18185) logged in. Now: 2 clients, [ Alice(127.0.0.1:18183), Bob(127.0.0.1:18185) ] [Chat::Server] @Bob is asking for symmetric key. @Alice will help!</pre>	<pre>\$ node dist/c1.js client [Chat::Client] Type the Address [127.0.0.1]: [Chat::Client] Type the Port [5535]: 5535 [Chat::Client] Type your nickname: Alice [Chat::Client] Connected at 127.0.0.1:5535 [Chat::Client] Logged as @Alice with success! [Chat::Client] [Alice]: [Chat::Client] Generated SymmetricKey: a53d473ac7f6b8b474775a86 6058bca7f52fede0182d7f8b40254d4c8c2 [Chat::Client] Bob: hi [Chat::Client] [Alice]: Hello [Chat::Client] [Alice]:</pre>	<pre>127.0.0.1:5535(127.0.0.1:18185) (Bob): \$ node dist/c1.js client [Chat::Client] Type the Address [127.0.0.1]: [Chat::Client] Type the Port [5535]: 5535 [Chat::Client] Type your nickname: Bob [Chat::Client] Connected at 127.0.0.1:5535 [Chat::Client] Logged as @Bob with success! [Chat::Client] [Bob]: [Chat::Client] HandShaked SymmetricKey: a53d 473ac7f6b8b474775a86205c9a7f52fede0182d7f8b40254d4c8c2 [Chat::Client] [Bob]: hi [Chat::Client] Alice: Hello</pre>

### 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

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 5535

No.	Time	Source	Destination	Protocol	Length	Info
4073...	17029.412136	127.0.0.1	127.0.0.1	TCP	44	5535 → 10185 [ACK] Seq=48 Ack=511 Win=2619648 Len=0
4073...	17029.412610	127.0.0.1	127.0.0.1	TCP	545	5535 → 10183 [PSH, ACK] Seq=62 Ack=513 Win=2619648 Len=501
4073...	17029.412641	127.0.0.1	127.0.0.1	TCP	44	10183 → 5535 [ACK] Seq=513 Ack=563 Win=2619136 Len=0
4073...	17029.413522	127.0.0.1	127.0.0.1	TCP	600	10183 → 5535 [PSH, ACK] Seq=513 Ack=563 Win=2619136 Len=556
4073...	17029.413550	127.0.0.1	127.0.0.1	TCP	44	5535 → 10183 [ACK] Seq=563 Ack=1069 Win=2619136 Len=0
4073...	17029.413765	127.0.0.1	127.0.0.1	TCP	600	5535 → 10185 [PSH, ACK] Seq=48 Ack=511 Win=2619648 Len=556
4073...	17029.413782	127.0.0.1	127.0.0.1	TCP	44	10185 → 5535 [ACK] Seq=511 Ack=604 Win=2619136 Len=0
4077...	17063.482319	127.0.0.1	127.0.0.1	TCP	132	10185 → 5535 [PSH, ACK] Seq=511 Ack=604 Win=2619136 Len=88
4077...	17063.482344	127.0.0.1	127.0.0.1	TCP	44	5535 → 10185 [ACK] Seq=604 Ack=599 Win=2619648 Len=0
4077...	17063.482624	127.0.0.1	127.0.0.1	TCP	132	5535 → 10183 [PSH, ACK] Seq=563 Ack=1069 Win=2619136 Len=88
4077...	17063.482645	127.0.0.1	127.0.0.1	TCP	44	10183 → 5535 [ACK] Seq=1069 Ack=651 Win=2619136 Len=0
4104...	17172.920774	127.0.0.1	127.0.0.1	TCP	142	10183 → 5535 [PSH, ACK] Seq=1069 Ack=651 Win=2619136 Len=98
4104...	17172.920799	127.0.0.1	127.0.0.1	TCP	44	5535 → 10183 [ACK] Seq=651 Ack=1167 Win=2618880 Len=0
4104...	17172.920993	127.0.0.1	127.0.0.1	TCP	142	5535 → 10185 [PSH, ACK] Seq=604 Ack=599 Win=2619648 Len=98
4104...	17172.921012	127.0.0.1	127.0.0.1	TCP	44	10185 → 5535 [ACK] Seq=599 Ack=702 Win=2618880 Len=0

<

> Frame 407772: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface \Device\NPF\_{Loopback}, id 0

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 10185, Dst Port: 5535, Seq: 511, Ack: 604, Len: 88

> Data (88 bytes)

Data: 53454e443d6165313039373564653231366262613463386337363133623a376130303265...

[Length: 88]

0000	02 00 00 00 45 00 00 80	6a f5 40 00 80 06 00 00	....E... j...@...
0010	7f 00 00 01 7f 00 00 01	27 c9 15 9f e2 11 80 99	.....^.....
0020	48 f1 e0 eb 50 18 27 f7	ed 8b 00 00 53 45 4e 44	H...P... SEND
0030	3d 61 65 31 30 39 37 35	64 65 32 31 36 62 62 61	=ae10975 de216bba
0040	34 63 38 63 37 36 31 33	62 3a 37 61 30 30 32 65	4c8c7613 b:7a002e
0050	64 38 66 36 35 61 37 31	35 61 37 31 65 62 36 34	d8f65a71 5a71eb64
0060	65 62 62 33 63 39 63 66	62 62 31 39 37 33 62 62	ebb3c9cf bb1973bb
0070	61 62 64 34 61 62 66 32	33 63 34 63 35 39 36 35	abd4abf2 3c4c5965
0080	65 63 65 64		eced

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.