

# RELATÓRIO — Sistema de Comunicação Segura com RSA + AES (CBC/CTR) sobre TCP

Francisco Gustavo Martins de Sousa - 511779

## 1. Introdução

Este trabalho apresenta o desenvolvimento de um sistema simples de comunicação segura baseado em sockets TCP e criptografia ponta a ponta.

O objetivo é demonstrar, na prática, como técnicas criptográficas modernas podem ser aplicadas para garantir confidencialidade e integridade em um ambiente de rede inseguro.

O sistema implementado utiliza:

- **RSA-OAEP** para troca de chaves de sessão
- **AES-256** nos modos **CBC** e **CTR**
- **TCP** como camada de transporte
- **Servidor store-and-forward** sem acesso ao conteúdo das mensagens
- **Cientes CLI**

## 2. Arquitetura Geral do Sistema

O sistema é composto por três principais módulos:

### 1. Servidor TCP

- Atua apenas como *roteador de mensagens*.
- Não lê plaintext.
- Armazena chaves públicas temporariamente para pareamento entre clientes.

### 2. Clientes

Cada cliente executa:

- Geração de chave **RSA** (2048 bits)
- Envio/recebimento de **chaves públicas**
- Geração de chave de sessão **AES (256 bits)**
- Troca de chaves cifradas via **RSA-OAEP**
- Derivação da chave final compartilhada
- Criptografia e descriptografia de mensagens (**CBC** ou **CTR**)

- Modo **DEBUG** opcional para análise

### 3. Módulo de Criptografia

- `cbc.py` — implementação manual do modo CBC usando AES básico
- `ctr.py` — implementação manual do modo CTR
- `key_exchange.py` — geração RSA, pacote OAEP, XOR final das chaves

## 3. Fluxo de Troca de Chaves

O protocolo criado segue estas etapas:

### 1. Cada cliente gera seu par RSA localmente

- 2048 bits
- Suporte a OAEP
- Chaves permanecem apenas localmente

### 2. Troca das chaves públicas via servidor

- O servidor apenas repassa

### 3. Cada cliente gera uma chave AES-256 aleatória

`aes_key_local = 32 bytes`

### 4. Cada lado cifra sua chave AES usando a chave pública do peer

Utilizando RSA-OAEP:

```
encrypted_key = RSA_OAEP(peer_pubkey).encrypt(aes_key_local)
```

### 5. Cada lado recebe a chave AES cifrada do peer e decifra com sua chave privada

### 6. Chave final é derivada via XOR

```
aes_final[i] = aes_local[i] XOR aes_peer[i]
```

Esse método garante:

- Chave final única
- Nenhum cliente sozinho controla a chave final
- Servidor não tem acesso a nenhuma parte das chaves

## Validando Fluxo

Para fins de validação e análise do funcionamento interno da criptografia, foi implementado um modo **DEBUG** acessível via o argumento `--debug` ao executar o cliente, por exemplo:

```
python -m client.client --user alice --peer bob --mode cbc --debug
```

Esse modo não altera a segurança do sistema, pois toda a informação exibida permanece apenas no terminal local do usuário. Ele serve exclusivamente para fins de auditoria e permite visualizar a geração da chave AES local, a chave recebida do peer, a derivação da chave final via XOR, além do IV/nonce e do ciphertext produzido em cada mensagem enviada.

**Alice em debug:**

- AES-LOCAL
- RSA-SEND
- RSA-RECV
- AES-PEER
- AES-FINAL

```

gusta@gustavo MINGW64 /d/estudos/2025.2/Criptografia/trabalho1/secure-chat
$ python -m client.client --user alice --peer bob --mode cbc --debug
[+] Iniciando cliente como 'alice', destino = 'bob', modo = CBC
[+] Conectado ao servidor.
[+] Par RSA gerado.
[+] Chave pública enviada. Aguardando chave do peer...
[+] Chave pública do peer recebida.
[DEBUG][AES-LOCAL ] 198e013687bddb8574b8e521033384d59a89775e8161daccbc65604a5ea092c3
[+] Chave AES de sessão gerada.
[DEBUG][RSA-SEND ] vWNVV1C4PiaZbqBpkwUVccYU0NsaaBI1NvEsVsINZ4YEP+V1qbwAe1bVG0sIWoCK82MDZzsV1wCABDr6X
Vku1lK0igPLzIwCUhTP5FXNAOP6EX7MNJJAPVvYZhFGWyy4ZC2/avQj4UPEH6nErzN1svxwX6HkRbw+i1mY+ZNmxGFSTRZ5vSXI8xAH
dzpjggnwJLZVeTxa0PyztFqWP6jqHBw7szqcMBCdv7q+zg3ATXJkYDfz/moQ90ZM737o072tGW8k6izS44rOVDCrga97RsNTNNxGUaa
ojqGjJwkgMdu/1zavhgwqnLmhlkby6dhcV0qTJ3FSBrOn95IBqsCOaw==
[+] Chave de sessão enviada ao peer.
[+] Aguardando chave de sessão do peer...
[DEBUG][RSA-RECV ] pr+e8iFz08c00Dghpo1jkIUs86eVH8WR1RPJkS8K2+iDpS4/GouQ+j1m4nPeq3iZ/spM4w2sH5f0fgaZv
pd7of4JrpsILu66rdv5X/zoB1lFZ7yJZcTA/6HWVCp7aFFaGRmy6ih04juuwwHcA/RzZf5SG/+K9qjHJ7DSzC47d2uv1ui1ukyVfcUZ
Q2uT33oQpn5dxhMzAyGnh9dVJaXJwSDRrWlk4Jvy5nvJBoVFYisE4sFoercN9Lp9Y0fVzCYqvqyQfy2jDcAY/ZI00a4hId2eSIJDVdI
VDTvW8X1jiIWNIE94ttjnuQ3LstEvH1Qod5nGj29h7mDyYasmE2scw==
[DEBUG][AES-PEER ] ec7b2227a9c6dddf97684f9958c10fa3dcade1f7d06abe6fc3b6a03d1aed70da
[DEBUG][AES-FINAL ] f5f523112e7b065ae3d0aab85bf28b76462496a9510b64a37fd3c077444de219
[+] Chave final compartilhada derivada com sucesso.

[CHAT ATIVO] Digite mensagens para enviar.
> >

```

### Bob em debug:

- AES-LOCAL diferente
- AES-FINAL igual à da Alice

```

gusta@gustavo MINGW64 /d/estudos/2025.2/Criptografia/trabalho1/secure-chat
$ python -m client.client --user bob --peer alice --mode cbc --debug
[+] Iniciando cliente como 'bob', destino = 'alice', modo = CBC
[+] Conectado ao servidor.
[+] Par RSA gerado.
[+] Chave pública enviada. Aguardando chave do peer...
[+] Chave pública do peer recebida.
[DEBUG][AES-LOCAL ] ec7b2227a9c6dddf97684f9958c10fa3dcade1f7d06abe6fc3b6a03d1aed70da
[+] Chave AES de sessão gerada.
[DEBUG][RSA-SEND ] pr+e8iFz08c00Dghpo1jkIUs86eVH8WR1RPJkS8K2+iDpS4/GouQ+j1m4nPeq3iZ/spM4w2sH5f0fgaZv
pd7of4JrpsILu66rdv5X/zoB1lFZ7yJZcTA/6HWVCp7aFFaGRmy6ih04juuwwHcA/RzZf5SG/+K9qjHJ7DSzC47d2uv1ui1ukyVfcUZ
Q2uT33oQpn5dxhMzAyGnh9dVJaXJwSDRrWlk4Jvy5nvJBoVFYisE4sFoercN9Lp9Y0fVzCYqvqyQfy2jDcAY/ZI00a4hId2eSIJDVdI
VDTvW8X1jiIWNIE94ttjnuQ3LstEvH1Qod5nGj29h7mDyYasmE2scw==
[+] Chave de sessão enviada ao peer.
[+] Aguardando chave de sessão do peer...
[DEBUG][RSA-RECV ] vWNVV1C4PiaZbqBpkwUVccYU0NsaaBI1NvEsVsINZ4YEP+V1qbwAe1bVG0sIWoCK82MDZzsV1wCABDr6X
Vku1lK0igPLzIwCUhTP5FXNAOP6EX7MNJJAPVvYZhFGWyy4ZC2/avQj4UPEH6nErzN1svxwX6HkRbw+i1mY+ZNmxGFSTRZ5vSXI8xAH
dzpjggnwJLZVeTxa0PyztFqWP6jqHBw7szqcMBCdv7q+zg3ATXJkYDfz/moQ90ZM737o072tGW8k6izS44rOVDCrga97RsNTNNxGUaa
ojqGjJwkgMdu/1zavhgwqnLmhlkby6dhcV0qTJ3FSBrOn95IBqsCOaw==
[DEBUG][AES-PEER ] 198e013687bddb8574b8e521033384d59a89775e8161daccbc65604a5ea092c3
[DEBUG][AES-FINAL ] f5f523112e7b065ae3d0aab85bf28b76462496a9510b64a37fd3c077444de219
[+] Chave final compartilhada derivada com sucesso.

[CHAT ATIVO] Digite mensagens para enviar.
> >

```

*Servidor mostrando recebimento das public keys sem conteúdo sensível.*

```
gusta@gustavo MINGW64 /d/estudos/2025.2/Criptografia/trabalho1/secure-chat/serve
r
$ python server.py
Servidor iniciando em 0.0.0.0:9000 ...
[OK] Servidor aguardando conexões.

[+] Conexão estabelecida com ('127.0.0.1', 49277)
[+] Cliente registrado: alice
[pubkey] Recebida chave pública de alice
[info] bob ainda não disponível. Chave salva.
[+] Conexão estabelecida com ('127.0.0.1', 62032)
[+] Cliente registrado: bob
[pubkey] Recebida chave pública de bob
[relay:pubkey] Enviando chave de bob para alice
[relay:pubkey] Enviando chave de alice para bob
[relay:session_key] alice -> bob
[relay:session_key] bob -> alice
```

## 4. Modos de Operação AES Implementados

O sistema suporta dois modos de operação:

### AES-CBC (Cipher Block Chaining)

- Requer IV aleatório de 16 bytes
- Necessita de padding (PKCS#7)
- Cada bloco depende do anterior
- Repetições no plaintext não aparecem no ciphertext

### AES-CTR (Counter Mode)

- Funciona como stream cipher
- Usa nonce + contador
- Não precisa de padding
- Muito rápido e paralelo

## 5. Funcionamento do Chat Criptografado

Após a chave final AES ser derivada:

## 1. O cliente cifra cada mensagem com AES-FINAL

- CBC → IV novo a cada mensagem
- CTR → nonce novo a cada mensagem

## 2. O servidor recebe somente:

```
{  
  "mode": "cbc",  
  "ciphertext": "base64...",  
  "iv": "base64..."  
}
```

Nada de plaintext.

## 3. O cliente destino descriptografa localmente.

Cliente mostrando:

- PLAINTEXT
- IV
- CIPHERTEXT

```
[CHAT ATIVO] Digite mensagens para enviar.  
> > ola bob  
[DEBUG][PLAINTEXT    ] ola bob  
[DEBUG][IV           ] 272af05c7bac65836df7ddc74c2a47be  
[DEBUG][CIPHERTEXT   ] 9ighTtS35eiwyR6GN15iHg==  
> [DEBUG][IV           ] 782b4622cbfcf12f836f35a9813fa9b4  
[DEBUG][CIPHERTEXT   ] 8PzMECfinh8iIGiA/dAKrw==  
[DEBUG][PLAINTEXT    ] ola alice  
  
[bob] ola alice  
> |
```

Servidor:

```
[relay:session_key] alice -> bob
[relay:session_key] bob -> alice
[relay:chat] alice -> bob
[relay:chat] bob -> alice
```

Cliente destino decodificando a mesma mensagem.

```
[CHAT ATIVO] Digite mensagens para enviar.
> > [DEBUG][IV          ] 272af05c7bac65836df7ddc74c2a47be
[DEBUG][CIPHERTEXT    ] 9ighTtS35eiwyR6GN15iHg==
[DEBUG][PLAINTEXT     ] ola bob

[alice] ola bob
> ola alice
[DEBUG][PLAINTEXT     ] ola alice
[DEBUG][IV          ] 782b4622cbfcf12f836f35a9813fa9b4
[DEBUG][CIPHERTEXT    ] 8PzMECfinh8iIGiA/dAKrw==
```

## 6. Como Rodar o Projeto

Para executar o sistema completo, recomenda-se utilizar **três abas/terminais** simultaneamente:

1. **Uma aba para o servidor**
2. **Duas abas para os clientes** (por exemplo, Alice e Bob)

Antes de iniciar, certifique-se de instalar a dependência necessária:

```
pip install pycryptodome
```

### 1. Iniciar o Servidor

Na pasta `server/`, execute:

```
python server.py
```

O servidor iniciará ouvindo na porta padrão **9000** e aguardará conexões dos clientes.

## 2. Iniciar o Cliente Alice

Em outra aba, na pasta raiz do projeto:

```
python -m client.client --user alice --peer bob --mode cbc
```

Parâmetros obrigatórios:

- `--user` → nome do usuário local
- `--peer` → destinatário esperado
- `--mode` → modo de operação criptográfica (`cbc` ou `ctr`)

## 3. Iniciar o Cliente Bob

Na terceira aba:

```
python -m client.client --user bob --peer alice --mode cbc
```

A troca de chaves será feita automaticamente, seguida pelo início da comunicação segura.

## 4. Ativar modo DEBUG

O modo DEBUG exibe informações internas para validação do funcionamento (chaves, IV, ciphertext etc.).

Basta adicionar o parâmetro:

```
--debug
```

Exemplo:

```
python -m client.client --user alice --peer bob --mode cbc --debug
```

# 7. Análise de Segurança

### Pontos fortes

- Criptografia ponta a ponta verdadeira
- RSA-OAEP evita ataques clássicos



- AES-256 é seguro no contexto atual
- Cada sessão gera novas chaves
- Derivação via XOR impede controle unilateral
- Servidor nunca vê plaintext
- CBC e CTR implementados manualmente

## 8. Conclusão

O projeto atingiu o objetivo proposto:

Foi implementado um sistema funcional de comunicação segura utilizando criptografia moderna, de forma clara, modular e didática. A troca de chaves RSA, derivação de chave AES, operação de criptografia com CBC/CTR e transporte via TCP foram validados com testes e logs de debug.