🎯 Objetivo

Implementar uma aplicação de **mensageria segura multi-cliente**, onde vários clientes se conectam a um servidor central e trocam mensagens de forma protegida. O servidor gerencia conexões, valida mensagens e as encaminha aos destinatários corretos. O sistema deve garantir **confidencialidade, integridade, autenticidade e sigilo perfeito**.

🔐 Requisitos de Segurança

1. **Confidencialidade**
   - As mensagens devem ser ilegíveis para terceiros.
   - **Mecanismo:** AES-128-GCM (AEAD).
2. **Integridade**
   - Alterações não autorizadas devem ser detectadas.
   - **Mecanismo:** Tag de autenticação do AES-GCM.
3. **Autenticidade**
   - O servidor deve ser autenticado pelos clientes.
   - **Mecanismo:** Certificado RSA autoassinado, usado para assinar a chave efêmera ECDHE.
4. **Sigilo Perfeito (Forward Secrecy)**
   - Mesmo que a chave RSA seja comprometida no futuro, as sessões passadas permanecem seguras.
   - **Mecanismo:** ECDHE (Elliptic Curve Diffie-Hellman Ephemeral).

🔑 Requisitos de Geração de Chaves

- **ECDHE:**
  - Cada cliente gera um par efêmero de chaves e envia sua chave pública ao servidor.
  - O servidor gera seu par efêmero e envia sua chave pública, assinada com RSA.
- **Assinatura RSA:**
  - O servidor assina `pk_S || client_id || transcript || salt` com sua chave privada RSA.
  - O cliente valida a assinatura usando o certificado autoassinado do servidor.
- **HKDF (TLS 1.3):**

- O segredo compartilhado ZZ do ECDHE é usado como **IKM (Input Keying Material)**.
- O servidor gera um **salt aleatório** e envia ao cliente.
- O HKDF é aplicado em duas fases:
  - **HKDF-Extract:** `PRK = HMAC(salt, Z)`
  - **HKDF-Expand:** gera chaves distintas com labels:
    - `Key_c2s = HKDF-Expand(PRK, "c2s", 16)`
    - `Key_s2c = HKDF-Expand(PRK, "s2c", 16)`

📦 Estrutura da Mensagem

```
[nonce (12B)] + [sender_id (16B)] + [recipient_id (16B)] + [seq_no (8B)] + [ciphertext+tag]
```

- **AAD (Associated Data):** `sender_id | recipient_id | seq_no`
- **Proteção:** AES-128-GCM garante confidencialidade, integridade e autenticidade.
- **Nonce:** deve ser único por mensagem e por direção.
- **Seq_no:** contador monotônico para evitar ataques de replay.

🗂️ Estrutura de Dados no Servidor (Exemplo em Python)

```
sessions = {
   client_id: {
      "writer": objeto StreamWriter,
      "key_c2s": chave AES-128-GCM (cliente→servidor),
      "key_s2c": chave AES-128-GCM (servidor→cliente),
      "seq_recv": último seq_no recebido (para replay detection),
      "seq_send": próximo seq_no a enviar (nonces únicos),
      "salt": valor usado na derivação HKDF
   },
   ...
}
```

📊 Fluxo do Protocolo (Visão Geral)

sequenceDiagram participant Cliente_A participant Servidor participant Cliente_B %% Handshake Cliente A Cliente_A->>Servidor: 1) Envia pk_C_A (ECDHE efêmero) Servidor->>Cliente_A: 2) Envia pk_S (ECDHE efêmero) + server.crt + assinatura RSA Cliente_A->>Cliente_A: 3) Verifica assinatura RSA usando server.crt (local) Note over Cliente_A,Servidor: Ambos calculam Z_A = ECDH(sk_local, pk_peer) Servidor->>Cliente_A: 4) Envia salt_A Note over Cliente_A: Deriva Key_c2s_A / Key_s2c_A via HKDF (TLS 1.3) Note over Servidor: Deriva Key_c2s_A / Key_s2c_A via HKDF (TLS 1.3) Servidor->>Servidor: Registra sessão {client_id=A, keys, seq counters} %% Handshake Cliente B Cliente_B->>Servidor: 5) Envia pk_C_B (ECDHE efêmero) Servidor->>Cliente_B: 6) Envia pk_S (ECDHE efêmero) + server.crt + assinatura RSA Cliente_B->>Cliente_B: 7) Verifica assinatura RSA usando server.crt (local) Note over Cliente_B,Servidor: Ambos calculam Z_B = ECDH(sk_local, pk_peer) Servidor->>Cliente_B: 8) Envia salt_B Note over Cliente_B: Deriva Key_c2s_B / Key_s2c_B via HKDF (TLS 1.3) Note over Servidor: Deriva Key_c2s_B / Key_s2c_B via HKDF (TLS 1.3) Servidor->>Servidor: Registra sessão {client_id=B, keys, seq counters} %% Mensagem de A para B Cliente_A->>Servidor: 9) Dados cifrados (AES-GCM com Key_c2s_A) + nonce + tag Note over Servidor: Decifra com Key_c2s_A, valida tag GCM Servidor->>Cliente_B: 10) Re-cifra com Key_s2c_B, envia pacote Note over Cliente_B: Decifra com Key_s2c_B, valida tag GCM e exibe mensagem

🔄 Fluxo Passo a Passo

1. Inicialização

- Cliente abre conexão TCP com o servidor.
- Servidor aguarda múltiplas conexões simultâneas.

2. Handshake

- **Cliente envia:** `client_id` + `pk_C`.
- **Servidor responde:** `pk_S` + `server.crt` + assinatura RSA + `salt`.
- **Cliente valida:** assinatura RSA com certificado pinado.
- **Ambos calculam:** segredo compartilhado $Z = ECDH(sk_local, pk_peer)$Z = ECDH(sklocal, pkpeer).
- **Ambos derivam:** chaves direcionais via HKDF (TLS 1.3).
- **Servidor registra:** sessão na tabela `sessions`.

3. Troca de Mensagens

- **Cliente → Servidor:** cifra mensagem com `Key_c2s`, envia frame com nonce, IDs, seq_no e

ciphertext.

- **Servidor:** decifra com `Key_c2s`, valida tag GCM e replay, re-cifra com `Key_s2c` do destinatário.
- **Servidor → Cliente Destinatário:** envia novo frame cifrado.
- **Cliente Destinatário:** decifra com `Key_s2c`, valida tag GCM e seq_no, exibe mensagem.

4. Encerramento

- Cliente desconecta → servidor remove entrada da tabela `sessions`.
- Nonces e seq_no não podem ser reutilizados.
- Opcional: rotação de chaves após N mensagens ou T minutos.

5. Garantias

- **Confidencialidade:** AES-GCM.
- **Integridade:** tag GCM.
- **Autenticidade:** certificado RSA.
- **Sigilo perfeito:** ECDHE.
- **Anti-replay:** seq_no monotônico.

📤 Entrega

1. O trabalho deve ser feito em **grupos de até 3 alunos**.
2. Pode ser implementado em qualquer linguagem de programação (Python, Java, C/C++, Go, Rust, etc.).
3. Cada grupo deverá enviar pelo **Moodle**, até o dia **21/01/2025**:
   - O **código-fonte** do trabalho (ou link para repositório GitHub).
   - Um documento descrevendo como rodar a aplicação (pode ser um **README.md** no GitHub).
   - Um **vídeo curto** (máximo 20 minutos) onde todos os membros do grupo apresentam parte do trabalho.

✅ Checklist para o Vídeo

- [ ] Apresentação dos integrantes (cada aluno fala e apresenta parte do trabalho).
- [ ] Explicação do protocolo (handshake ECDHE + RSA + HKDF e troca de mensagens

AES-GCM).

- [ ] Demonstração prática: inicialização do servidor, conexão de múltiplos clientes, envio de mensagens.
- [ ] Validação de segurança: mostrar que apenas o destinatário consegue decifrar.
- [ ] Estrutura de dados do servidor: explicar como sessões, chaves e contadores são gerenciados.
- [ ] Explicação do código: cada aluno apresenta trechos diferentes (handshake, derivação de chaves, criptografia, roteamento).
- [ ] Encerramento: resumo das garantias de segurança alcançadas.