Plano de Projeto: Sistema Seguro de Comunicação

Sprint 1: Fundação da Segurança - Autenticação de Usuário

Objetivo:
Implementar autenticação segura de usuários com hash de senhas.

Tarefas:

- Implementar módulo password_hashing.py com bcrypt ou Argon2.

- Criar user_auth.py com funções para registro, login e verificação de senha.

- Armazenamento temporário de credenciais (pode ser em JSON ou banco simples).

- Testes unitários em test_authentication.py.

Motivo:
Todo sistema seguro precisa autenticar seus usuários antes de enviar/receber mensagens.

Sprint 2: Criptografia Simétrica e Integridade

Objetivo:
Implementar criptografia de mensagens com AES e garantir integridade com HMAC.

Tarefas:

- Finalizar aes_encryption.py (usando CBC com padding ou GCM).

- Implementar hmac_verification.py usando HMAC-SHA256.

- Criar testes de criptografia e integridade (test_crypto.py).

Motivo:
Antes de enviar mensagens, precisamos garantir que ninguém mais as leia ou modifique.

Sprint 3: Troca Segura de Chaves e Assinaturas

Objetivo:
Proteger o canal de comunicação contra espionagem e falsificação.

Tarefas:

- Implementar key_exchange.py com Diffie-Hellman (ou ECDH).

- Desenvolver rsa_signatures.py para assinatura e verificação de mensagens.

- Criar random_generator.py com verificação do PRNG usado para IVs e chaves.

- Testes de segurança e geração de logs de operação.

Motivo:
Evita ataques do tipo Man-in-the-Middle e autentica a origem das mensagens.

Sprint 4: Canal de Comunicação Ponto-a-Ponto

Objetivo:
Simular a troca de mensagens segura entre dois clientes (Sender e Receiver).

Tarefas:

- Implementar channel.py usando sockets TCP/IP com SSL opcional.

- Criar sender.py e receiver.py que usam os módulos de criptografia e autenticação.

- Proteger o canal com as chaves trocadas e verificar HMACs.

Motivo:
Agora temos tudo para testar a aplicação real de ponta a ponta.

Sprint 5: Testes, Certificados e Simulação de Ataques

Objetivo:
Garantir que o sistema seja seguro, mesmo diante de tentativas de ataque.

Tarefas:

- Gerar certificados digitais autoassinados (certs/).

- Simular ataque Man-in-the-Middle (alterar canal).

- Testar brute-force nas senhas e ver comportamento do sistema.

- Criar relatórios finais de testes.

Motivo:
Validar se o sistema resiste a cenários reais de ameaça.