Trabalho Prático 1: Construindo um Sistema de Mensagens Seguras em Python

Disciplina: Segurança da Informação - Cursos Redes de Computadores

Prof. Michel Sales

Objetivo: Este exercício desafia você a construir uma aplicação de troca de mensagens segura em **Python** usando **sockets**, incorporando princípios criptográficos fundamentais para garantir confidencialidade, integridade e autenticidade.

Cenário: Você desenvolverá dois programas: um `servidor.py` e um `cliente.py`. O cliente enviará uma mensagem ao servidor, e o servidor a receberá e validará. Toda a comunicação das mensagens reais deverá ser protegida.

Requisitos de Segurança

—

1. **Confidencialidade:** A mensagem deve ser ilegível para um bisbilhoteiro.
   - **Mecanismo:** Criptografia simétrica usando **AES (Advanced Encryption Standard)** no modo CBC (Cipher Block Chaining).
2. **Integridade:** A mensagem não deve ser modificada em trânsito sem ser detectada.
   - **Mecanismo:** Uso de um **código de autenticação de mensagem baseado em hash (HMAC)**.
3. **Autenticidade:** A origem da mensagem deve ser verificável (garantindo que a mensagem veio do cliente legítimo).
   - **Mecanismo:** O HMAC também servirá para autenticação da origem, pois apenas as partes que compartilham a chave do HMAC podem gerar e verificar o HMAC corretamente.

Requisitos de Geração de Chaves

—

1. **Chave de Troca DH:** Para estabelecer uma chave compartilhada entre cliente e servidor, utilize o algoritmo **Diffie-Hellman (DH)**. Esta chave DH será a "semente" para a derivação das chaves finais.

2. **Handshake com Assinatura Digital (ECDSA):** Para aumentar a segurança do protocolo de troca de chaves DH, esta extensão propõe o uso de **assinaturas digitais com Elliptic Curve Digital Signature Algorithm (ECDSA)** durante o handshake. O objetivo é garantir a **autenticidade das chaves públicas DH** trocadas entre cliente e servidor, prevenindo ataques do tipo *man-in-the-middle*. Para tanto:

    - Cada parte (cliente e servidor) possui um par de chaves ECDSA (privada e pública).
    - As chaves públicas estão publicadas em `https://github.com/USER.keys`.

3. **Derivação de Chaves para AES e HMAC:** Após a troca DH, a chave secreta compartilhada pelo DH não deve ser usada diretamente como chave de criptografia ou HMAC. Em vez disso, utilize o algoritmo **PBKDF2 (Password-Based Key Derivation Function 2)** para derivar duas chaves separadas e seguras a partir da chave DH:

    - Uma chave para a criptografia AES (`Key_AES`).
    - Uma chave para o HMAC (`Key_HMAC`).
    - **Salt:** O PBKDF2 requer um *salt*. Você pode gerar um *salt* aleatório e trocá-lo durante o handshake inicial (ou defini-lo como um valor fixo, embora menos seguro para aplicações reais).
    - **Iterações:** Escolha um número de iterações adequado para o PBKDF2 (ex: 100.000 ou mais).

Estrutura da Mensagem (Após o Handshake DH/PBKDF2)

— A mensagem enviada pelo cliente ao servidor deve ter a seguinte estrutura (ou similar):
`[HMAC_TAG] + [IV_AES] + [MENSAGEM_CRIPTOGRAFADA]`

- `HMAC_TAG`: A resultado do cálculo HMAC sobre a `IV_AES` e a `MENSAGEM_CRIPTOGRAFADA`.
- `IV_AES`: O Vetor de Inicialização (IV) usado na criptografia AES. Este IV deve ser único para cada mensagem e transmitido em texto claro (não precisa ser secreto, mas deve ser imprevisível).
- `MENSAGEM_CRIPTOGRAFADA`: A mensagem original criptografada com AES usando `Key_AES`.

Passos para Implementação

—

1. **Configuração Inicial:**

    - Definir os parâmetros públicos para o Diffie-Hellman (um número primo grande p e um gerador g). Estes podem ser hardcoded no seu programa para simplificação do exercício, mas em uma aplicação real seriam negociados ou viriam de uma fonte confiável.

2. **Handshake Diffie-Hellman:**

   - **Cliente:**
     - Gera par DH: $a, A = g^a \mod p$
     - Assina A + `username_cliente` com sua chave privada ECDSA: `sig_A = ECDSA_sign(A+username_cliente)`
     - Envia para o servidor: A, `sig_A`, `username_cliente`
   - **Servidor:**
     - Recebe A, `sig_A`, `username_cliente`
     - Baixa a chave pública do cliente de `https://github.com/username_cliente.keys`
     - Verifica `sig_A` com a chave pública
     - Gera par DH: $b, B = g^b \mod p$
     - Assina B + `username_servidor`: `sig_B = ECDSA_sign(B+username_servidor)`
     - Envia para o cliente: B, `sig_B`, `username_servidor`
     - Calcula a chave secreta compartilhada $S_{\text{servidor}} = A^b \pmod p$.
   - **Cliente:**
     - Recebe B, `sig_B`, `username_servidor`
     - Baixa a chave pública do servidor de `https://github.com/username_servidor.keys`
     - Verifica `sig_B` com a chave pública
     - Calcula a chave secreta compartilhada $S_{\text{cliente}} = B^a \pmod p$.
   - **Verificação:** Ambas as partes devem ter chegado ao mesmo valor S.

3. **Derivação de Chaves (PBKDF2):**

   - Ambas as partes usarão o valor S (do DH) como "password"para o PBKDF2.
   - Derivar `Key_AES` e `Key_HMAC` usando um *salt* (o mesmo para ambos os lados) e um número de iterações.
     - Exemplo: `Key_AES = PBKDF2(S, salt, iterations, length_AES_key)`
     - Exemplo: `Key_HMAC = PBKDF2(S, salt, iterations, length_HMAC_key)`

4. **Troca de Mensagens Seguras:**

   - **Cliente (Envio):**
     - Gera uma mensagem em texto claro.
     - Gera um IV aleatório para AES.
     - Criptografa a mensagem com `Key_AES` e o IV gerado.
     - Calcula o HMAC da `IV_AES` + `MENSAGEM_CRIPTOGRAFADA` usando `Key_HMAC`.
     - Concatena `HMAC_TAG`, `IV_AES` e `MENSAGEM_CRIPTOGRAFADA` e envia como um único pacote via socket.
   - **Servidor (Recebimento):**
     - Recebe o pacote da mensagem.
     - Separa o `HMAC_TAG`, `IV_AES` e `MENSAGEM_CRIPTOGRAFADA`.
     - Calcula o HMAC esperado da `IV_AES` + `MENSAGEM_CRIPTOGRAFADA` usando sua `Key_HMAC`.

- **Verifica Integridade e Autenticidade:** Compara o `HMAC_TAG` recebido com o HMAC esperado. Se não forem idênticos (em tempo constante, para evitar ataques de temporização), rejeita a mensagem.
- Se o HMAC for válido, descriptografa a `MENSAGEM_CRIPTOGRAFADA` usando `Key_AES` e `IV_AES`.
- Exibe a mensagem em texto claro.

Avaliação

—

Seu programa será avaliado com base na:

- Correta implementação dos protocolos DH, PBKDF2, AES e HMAC.
- Garantia da confidencialidade, integridade e autenticidade da mensagem.
- Capacidade do cliente de enviar e do servidor de receber e validar a mensagem.
- Organização e clareza do código.

Boa sorte!