SCRAM Protocol

SCRAM (Salted Challenge Response Authentication Mechanism) is a password-based authentication protocol designed to authenticate both the client and the server without ever sending the password in cleartext. The following points summarize how SCRAM achieves client authentication, server authentication, and where it becomes vulnerable if TLS is not used.

a. Client Authentication

The client proves knowledge of the password through the ClientProof.
This is the central element that authenticates the client.

1. The client derives a *SaltedPassword* using the salt and iteration count provided by the server.
2. From this value, it computes:
    - *ClientKey = HMAC(SaltedPassword, "Client Key")*
    - *StoredKey = H(ClientKey)*
3. The client then computes:
    - *ClientSignature = HMAC(StoredKey, AuthMessage)*
    - *ClientProof = ClientKey XOR ClientSignature*

The server verifies the proof by recomputing the signature and recovering *ClientKey.*
If the recovered key matches the stored hash, the client is authenticated.

In short:
Client authentication is provided by *ClientProof*, which proves that the client knows the password-derived key without revealing the password.

b. Server Authentication

SCRAM also authenticates the server using the *ServerSignature.*

1. The server derives:
    - *ServerKey = HMAC(SaltedPassword, "Server Key")*
2. It computes:
    - *ServerSignature = HMAC(ServerKey, AuthMessage)*
3. The client verifies this value using its own locally derived ServerKey.

If the signature matches, the client knows that the server holds the correct password database entry and is not an impostor.

In short:
Server authentication is provided by *ServerSignature*, which proves that the server possesses the correct password-derived secret.

c. Offline Dictionary Attacks Without TLS

SCRAM is secure against many attacks, but without TLS it becomes vulnerable to *offline dictionary attacks.*

During the exchange, the server sends:

- the salt
- the iteration count
- the server nonce

These values, together with the captured ClientProof and AuthMessage, allow an attacker to:

1. Guess a password from a dictionary
2. Compute the corresponding SaltedPassword
3. Derive ClientKey, StoredKey, and ClientSignature
4. Check whether the computed ClientProof matches the intercepted one

Because this process can be repeated offline and at high speed, SCRAM *requires TLS* to hide these values and prevent attackers from collecting the material needed for offline guessing.

In short:
Without TLS, the exposure of salt, iterations, and ClientProof enables offline dictionary attacks.