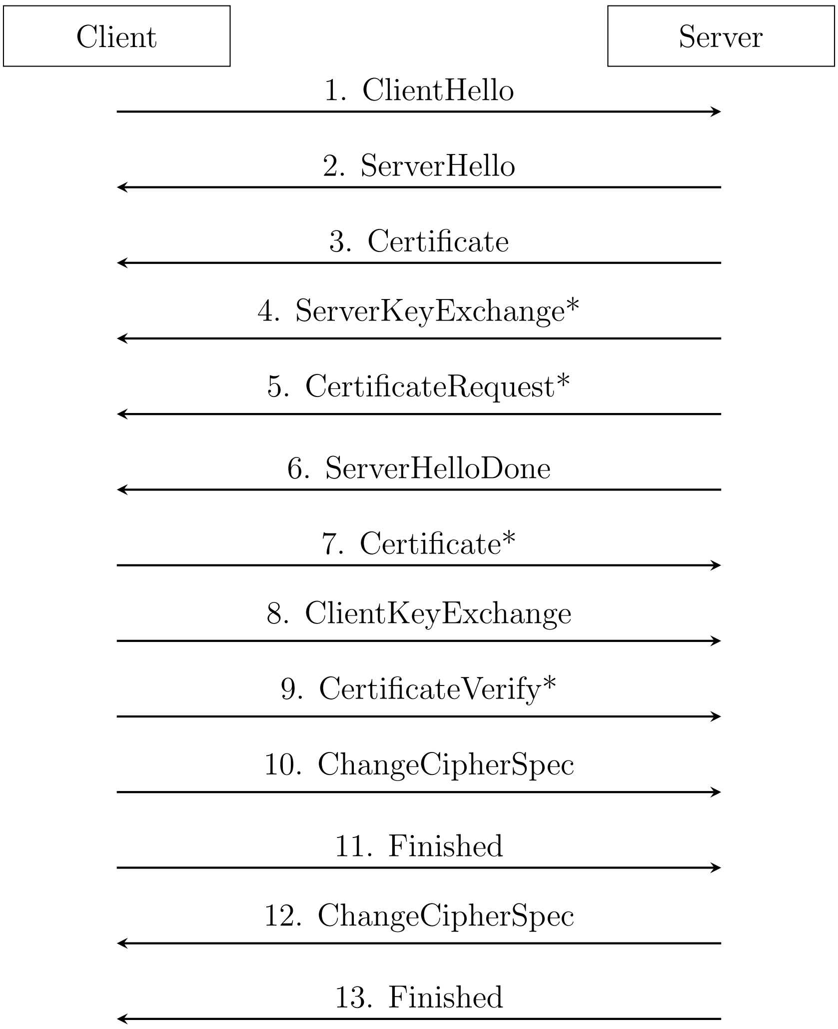
Proposal for Multi-Party Authorization via TLS Transport Layer Security Protocol

November 15, 2025

1 TLS 1.2 Protocol Overview

Transport Layer Security (TLS) 1.2 establishes a secure session through authentication, cipher negotiation, and key establishment. The handshake protocol consists of 13 messages exchanged between client and server:



* Optional or conditional messages

1.1 Message Descriptions

1. ClientHello: The client initiates the handshake with:
TLS version, 32-byte random, session ID, cipher suites list, compression methods, and extensions (SNI, ALPN, signature algorithms).

2. ServerHello: The server selects:
TLS 1.2, 32-byte server random, session ID, single cipher suite, compression method, and extension responses.

3. Certificate: X.509 certificate chain (server certificate + intermediate CAs). Client validates signature, validity, revocation status (CRL/OCSP), and hostname.

4. ServerKeyExchange (Optional): For DHE/ECDHE only. Contains DH parameters (p, g, $g^a \mod p$) or ECDH public key, signed with server's private key to prevent MITM.

5. CertificateRequest (Optional): If client authentication required. Specifies acceptable certificate types, trusted CAs, and signature algorithms.

6. ServerHelloDone: Indicates server completed its handshake phase.

7. Certificate (Optional): Client certificate chain if requested by server.

8. ClientKeyExchange: Key exchange material:

- **RSA**: 48-byte pre-master secret encrypted with server's public key
- **DHE/ECDHE**: Client's DH/ECDH public key value

9. CertificateVerify (Optional): If client sent certificate, signs hash of all handshake messages with client's private key.

1.2 Key Derivation

Both client and server compute:

$$\text{master\_secret} = \text{PRF}(\text{pre-master\_secret}, \text{“master secret”}, \text{ClientHello.random} + \text{ServerHello.random})$$

$$\text{key\_block} = \text{PRF}(\text{master\_secret}, \text{“key expansion”}, \text{ServerHello.random} + \text{ClientHello.random})$$

From the key block: client/server write MAC keys, encryption keys, and IVs (for block ciphers).

10. ChangeCipherSpec (Client): Activates encryption with newly negotiated keys.

11. Finished (Client): First encrypted message. Contains verify_data = PRF(master_secret, "client finished", Hash(handshake_messages)).

12. ChangeCipherSpec (Server): Server activates encryption.

13. Finished (Server): Encrypted message with verify_data = PRF(master_secret, "server finished", Hash(handshake_messages)). Handshake complete.

The standard TLS 1.2 key exchange uses RSA (no forward secrecy), DHE ($g^{ab} \mod p$), or ECDHE ($d_s \cdot Q_c = d_c \cdot Q_s$) to establish a pre-master secret, from which the master secret is derived using PRF. Session resumption reuses cached master secrets to abbreviate the handshake.

2 Proposal: Multi-Party Authorization in TLS

Current TLS implementations vest complete control of the server's private key in a single entity. This creates a single point of failure and trust. We propose modifying TLS to require authorization from multiple parties before establishing a secure session.

2.1 Proposed Approaches

We present two approaches to enforce multi-party authorization in TLS by modifying the key derivation mechanism:

2.1.1 Approach 1: Secret Sharing for Key Reconstruction

Mechanism: Use Shamir's Secret Sharing to split the server's private key among n parties.

Protocol Modification:

1. **Setup Phase:**
   - Server private key sk is split into n shares using (t, n)-threshold secret sharing
   - Each party P_i receives share s_i
   - Public key remains in certificate
2. **Modified Handshake (RSA Key Exchange):**
   - Client sends ClientHello
   - Server sends ServerHello, Certificate, ServerHelloDone
   - Client sends ClientKeyExchange with encrypted pre-master secret: $E_{pk}(PMS)$
   - **NEW:** To decrypt, at least t parties must cooperate:
     - Each party P_i contributes their share s_i
     - Reconstruct complete private key: $sk = \text{Reconstruct}(s_1, s_2, \ldots, s_t)$
     - Decrypt pre-master secret: $PMS = D_{sk}(E_{pk}(PMS))$
     - **Critical:** Complete key sk exists temporarily during decryption
   - Master secret derived from reconstructed PMS

Implementation Details: Shamir's Secret Sharing Scheme:
To split the server's RSA private key sk into n shares with threshold t:

1. **Share Generation:**
   - Choose random polynomial $f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{t-1}x^{t-1}$ where $a_0 = sk \mod p$ for large prime p
   - Generate shares: $s_i = f(i) \mod p$ for $i = 1, 2, \ldots, n$
   - Distribute share s_i to party P_i over secure channel
2. **Key Reconstruction (during handshake):**
   - Collect t shares from cooperating parties: $(i_1, s_{i_1}), (i_2, s_{i_2}), \ldots, (i_t, s_{i_t})$
   - Use Lagrange interpolation to recover $sk = f(0)$:

$$sk = \sum_{j=1}^{t} s_{i_j} \cdot \prod_{\substack{k=1 \\ k \neq j}}^{t} \frac{i_k}{i_k - i_j} \mod p$$

   - Decrypt PMS: $PMS = sk^d \mod N$ where d is RSA private exponent
   - Securely erase reconstructed sk after use
3. **Security Properties:**
   - Any $t-1$ or fewer shares reveal no information about sk (information-theoretic security)
   - Shares must be refreshed periodically to prevent gradual compromise
   - Communication between parties must be authenticated and encrypted

Example with $t = 3, n = 5$:
Suppose server's private key $sk = 12345$ (simplified), prime $p = 15485863$:

- Random polynomial: $f(x) = 12345 + 7891x + 2468x^2 \mod p$
- Generate shares:

$$\begin{aligned}
s_1 &= f(1) = 12345 + 7891 + 2468 = 22704 \mod p \\
s_2 &= f(2) = 12345 + 15782 + 9872 = 37999 \mod p \\
s_3 &= f(3) = 12345 + 23673 + 22212 = 58230 \mod p \\
s_4 &= f(4) = 12345 + 31564 + 39488 = 83397 \mod p \\
s_5 &= f(5) = 12345 + 39455 + 61700 = 113500 \mod p
\end{aligned}$$

- Any 3 shares can reconstruct sk, but 2 or fewer cannot
- To decrypt during handshake: collect shares from any 3 parties, reconstruct sk, decrypt PMS

2.1.2 Approach 2: Threshold Decryption of Master Secret

Mechanism: Use threshold encryption (e.g., ElGamal, Paillier) where decryption inherently requires multiple parties without ever reconstructing the private key.

Protocol Modification:

1. **Setup Phase:**
    - Generate threshold public key pk through distributed key generation (DKG)
    - Each party P_i holds private key share sk_i
    - pk cannot decrypt without t parties cooperating
    - Certificate contains threshold public key pk
2. **Modified Handshake:**
    - Standard ClientHello, ServerHello, Certificate exchange
    - Client sends ClientKeyExchange: $C = E_{pk}(PMS)$
    - **NEW:** Threshold decryption without key reconstruction:
        - Each party P_i computes decryption share: $d_i = D_{sk_i}(C)$
        - Shares include zero-knowledge proofs of correctness
        - Combiner aggregates shares: $PMS = \text{ThresholdDecrypt}(d_1, \ldots, d_t, C)$
        - **Key property:** Individual sk_i never combined; PMS computed from shares
    - Master secret derived: $MS = \text{PRF}(PMS, \text{label}, \text{randoms})$
    - Session keys derived from master secret
3. **Verification:**
    - Each party can publish proof that decryption share is correctly computed
    - Client or auditor verifies proofs before accepting PMS
    - Prevents malicious parties from contributing invalid shares

Advantages:

- Private key never exists in complete form (stronger security model)
- Individual compromise of $< t$ parties reveals nothing
- Zero-knowledge proofs ensure correctness
- Can support proactive secret sharing (refresh shares periodically)
- Natural fit for distributed systems and multi-organizational trust

Disadvantages:

- Requires changes to TLS cipher suite specifications
- More complex DKG setup protocol
- Computational overhead for threshold operations