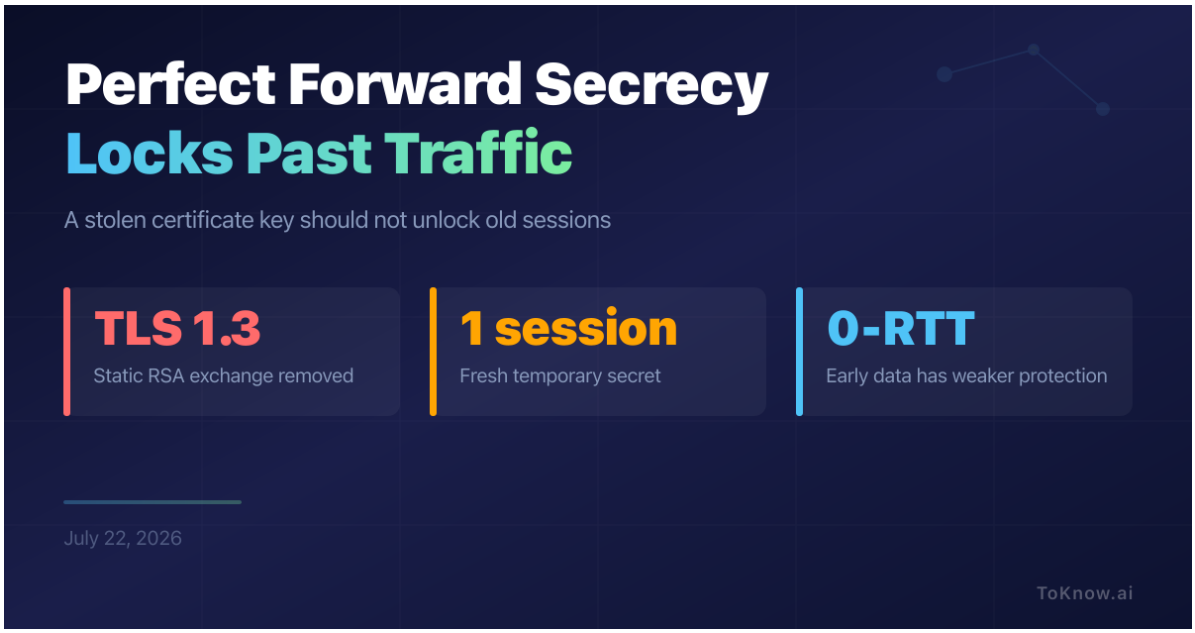
Perfect Forward Secrecy: A Stolen TLS Key Cannot Unlock Old Traffic

Kabui, Charles

2026-07-22

Read at ToKnow.ai



Forward secrecy limits what a stolen long-term TLS key can reveal. In a normal TLS 1.3 public-key handshake, the client and server create a temporary shared secret with ephemeral Diffie-Hellman, meaning they use fresh key material for that connection. The server's certificate key proves its identity but does not encrypt the session. TLS 1.3 removed static RSA and static Diffie-Hellman key exchange, which let a later private-key theft expose recorded traffic.

If someone records an HTTPS connection today and steals the certificate key next year, that key alone cannot reconstruct the deleted session secret. Resumed sessions keep this protection when a pre-shared key is combined with fresh `(EC)DHE`; PSK-only mode does not. 0-RTT early data also has weaker forward-secrecy guarantees.

Run `openssl s_client -brief -connect github.com:443 -servername github.com -tls1_3 </dev/null` to inspect a real connection. On July 22, 2026, it reported TLS 1.3, `TLS_AES_128_GCM_SHA256`, an ECDSA P-256 signature, and a temporary X25519 key. The signature shows how the server authenticated itself. The temporary key shows how both sides created the session secret. The cipher suite names the encryption and hash algorithms. An ECDSA certificate alone does not prove forward secrecy; an ephemeral key exchange does.

Forward secrecy protects recorded traffic only from a later long-term-key theft. It cannot protect plaintext logs, stored messages, a compromised browser or server, or session keys still in memory. An attacker controlling an endpoint during the connection can read plaintext. A live intermediary with a trusted certificate can terminate TLS and read the traffic too.

Sources:

- TLS 1.3 specification, RFC 8446
- Secure TLS recommendations, RFC 9325
- OWASP Transport Layer Security Cheat Sheet
- OpenSSL s_client documentation
