Engineer

# TLS

🤝 認証された暗号化接続を提供するプロトコル

## どのようなものか

Transport Layer Security、略してTLSは、クライアントとサーバーがアプリケーションのデータを盗聴や気付かれない改変から守り、通信相手を認証するためのプロトコルです。通常はクライアントがサーバーを認証し、クライアントの認証は任意です。製品のメニューや証明書の販売では、置き換えられたSecure Sockets Layerに由来するSSLという名前が今も使われます。RFC 7568は2015年6月にSSLの最後の版を廃止したため、今SSLと表示されているものは、内部ではTLSであるか、そうなっているべきです。

TLSは、アプリケーションプロトコルと、その下のトランスポートの間に置かれます。HTTPSはTLS接続の中でHTTPを運ぶもので、同じ層がメール送信、IMAP、データベース接続、Geminiも保護します。HTTP/3はさらに踏み込み、QUIC上で動きます。RFC 9001が説明するように、TCPの上に重ねるのではなく、TLS 1.3のハンドシェイクをトランスポート自体に組み込みます。

現在の版はTLS 1.3です。2026年7月公開のRFC 9846は、版番号を変えずに仕様を更新し、2018年8月のRFC 8446を置き換えます。通常の証明書ベースの接続は、ハンドシェイクから始まります。クライアントは、対応する版、暗号スイート、鍵共有情報を列挙したClientHelloを送ります。通常は、接続したいホスト名を示すServer Name Indication（SNI）も含めるため、一つのアドレスで複数のサイトを扱えます。サーバーが選択してServerHelloを返し、それに続くハンドシェイクのメッセージは、提示する証明書も含めて暗号化されます。証明書チェーンは、サーバーの鍵を、クライアントがすでに信頼する認証局へ結び付けます。クライアントがチェーン、名前、有効期間を調べ、サーバーはハンドシェイクに署名して、対応する秘密鍵を持つことを証明します。再開した接続では、以前に確立した事前共有鍵を使って認証することもできます。

1.3と1.2の主な違いは三つあります。通常の完全なハンドシェイクは、二往復ではなく一往復で済みますが、再試行によってもう一往復増えることがあります。公開鍵ベースの鍵確立は前方秘匿性を提供します。今日の通信を記録されても、翌年サーバーの長期秘密鍵が漏れたときに内容を読まれません。事前共有鍵だけを使うモードは例外で、前方秘匿性を提供しません。また、再訪するクライアントは、ハンドシェイク完了前の最初のメッセージで「早期データ」（early data）を送ることができます。0-RTTと呼ばれる方式ですが、RFCが明記する代償があります。early dataには前方秘匿性がなく、別の接続で再送されるリプレイからも保護されません。

概念を判断に生かす

「TLSとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 アカウント、パスワード、権限を整理し、業務に必要なアクセスだけを付与し、退職者の権限は当日に解除します。

**サービスを見る** お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 Linuxサーバーを守る1,000件のRBACルールを作成 — 1,000 セキュリティ上の穴を塞ぐアクセスルール

**Linuxサーバーを守る1,000件のRBACルールを作成** お問い合わせ →

## なぜ重要なのか

TLSを使うこと自体に議論の余地はありません。残る判断は、受け入れる版と、暗号化された接続をどこで終端するかです。

まず版を決めます。2021年3月のRFC 8996は、TLS 1.0と1.1を使用してはならないとしています。BCP 195にはRFC 9325と、その2026年7月の更新が含まれます。RFC 9852は、TLSを使う新しいプロトコルに1.3への対応と既定での使用を要求し、相互運用のために1.2を、既定ではない追加の選択肢として認めています。すべてのデプロイを1.3専用にする要件ではありません。RFC 10015は、TLS 1.2のRSA鍵交換と、有限体Diffie–Hellmanの鍵交換を、エフェメラルな有限体DHも含めて禁止しています。証明書のRSA署名は禁止していません。古いクライアントにも対応する必要がある既存サービスでは、1.3とともに残す1.2は、ECDHEとAES-GCMのような認証付き暗号を使い、安全に設定したものに限定します。1.3専用サービスは攻撃対象領域が小さく、設定も短くなりますが、最大でも1.2までしか対応しない古い電話、カード決済端末、古いJava実行環境を使う取引先の連携をすべて締め出します。数週間、実際に合意した版を記録し、先進的に聞こえるかではなく、その数値で判断します。

次に終端です。TLSが終わる場所に秘密鍵があり、そこからトラフィックは平文に戻ります。一般的な配置先は三つです。

- **CDNやクラウドのロードバランサー。** 事業者が鍵、または自分で発行した証明書を持ち、ハンドシェイクの負荷を引き受け、すべての要求を平文で見ます。そこからオリジンまでの区間にも独自のTLSが必要で、なければ暗号化されずにネットワークを通ります。
- **ホスト上のリバースプロキシ。** Nginx、Apache、HAProxy、Caddyなどです。一つのプロセスがマシン上の全サイトの鍵を持ち、背後のアプリケーションへ、ループバックやローカルソケットを通じて平文の要求を渡します。
- **アプリケーション自体。** 各サービスが自分の鍵を持ちます。サービス間通信や、クライアントも証明書を提示する相互TLSに適していますが、証明書を更新する場所は増えます。

多くのインフラでは、三つを組み合わせています。大事なのは、各区間について、平文がどこにあり、誰が読めるかを説明できることです。

## 実際の活用例

同社の2026年9月の公開サービス実装は、TLS終端に必要な資源の少なさを示しています。使ったのは、単一コア、メモリー512 MB、うち使用可能な約464 MB、ディスク10 GBのクラウドインスタンスでした。Webサーバーが七つのドメインのTLSを終端し、同じマシンでGitフォージ、Torのonionサービス、Geminiサーバー、Gopherサーバー、侵入元の遮断も動いていました。バックアップの自動化は別に実装されており、それだけで暗号化バックアップが有効だったとは確認できません。ホスト上での終端を最小限の形にした構成で、一つのWebサーバーが七つのサイトに応答し、暗号化のための追加マシンも追加サービスも不要でした。設計ではメモリーを制約の中心に置き、記録されたデプロイは464 MB内で動作していました。

同社が従う方針は、セキュリティのページに記載されています。すべての通信を暗号化し、証明書の発行と更新を自動化し、更新を監視します。その理由として、TLSが失敗する最も多い原因は、誰も監視していない証明書だと説明しています。以下は、更新が自動で動くようになっても残る失敗です。

## 問題が起きるところ

**ブラウザーはつながるのに、一部のクライアントでハンドシェイクが失敗する。** 両端に共通する版か暗号スイートがありません。サーバーを1.3専用や短いスイート一覧に絞ったため古いクライアントが提示する候補と一致しなくなったり、CBCスイートしか提供しないサーバーに、CBCを廃止したクライアントが接続したりします。警告は「handshake failure」や「protocol version」と示すだけで、詳しい事情は分かりません。外部から各版を試します。

```
openssl s_client -connect example.com:443 \
    -servername example.com -tls1_2 </dev/null
```

絞り込む前に、合意した版とスイートを記録します。クライアントがまだ1.2を必要とする既存サービスでは、移行を計画しながら、安全に設定したECDHEとAES-GCMの1.2を1.3と併用します。TLSを使う新規プロトコルでは1.3をサポートし、既定にします。1.2は、相互運用で必要な場合だけ有効にします。

**ブラウザーは証明書を信頼するが、スクリプト、curl、モバイルアプリは拒否する。** サーバーが、ルートにつなぐ中間証明書を送らず、自分の証明書だけを送っています。ブラウザーは以前見た中間証明書で補ったり、不足分を自分で取得したりすることが多い一方、ほとんどのTLSライブラリはどちらもせず、発行者不明と報告します。サーバーの証明書を先頭に、中間証明書を後に置いた完全なチェーンファイルを設定し、キャッシュを持たないクライアントで確認します。上の `openssl s_client` は、実際にサーバーが送ったチェーンを表示します。

**証明書の検証を「一時的に」止め、その設定で出荷する。** 自己署名証明書のテストサーバーや、通信に再署名する社内プロキシへの対処として、`verify=False` 、 `curl -k` 、 `InsecureSkipVerify` 、 `NODE_TLS_REJECT_UNAUTHORIZED=0` を追加してしまいます。接続は暗号化されたままですが、相手は応答した誰かです。途中の第三者が自分の証明書を提示して、すべて読むことができます。正しいCA証明書をクライアントの信頼ストアに入れるか、明示的に参照先を指定し、それらのフラグがあればCIを失敗させます。

**一部の訪問者に、別サイトの証明書が届く。** ClientHelloが、設定のないホストを指定したり、ホストを指定しなかったりすると、サーバーは既定の証明書を提示します。新しいドメインをサーバーブロック作成前にアドレスへ向けた場合や、監視プローブや古いクライアントがSNIを送らない場合に起きます。ブラウザーには名前の不一致が表示され、共有ホストでは、そこにあるほかのサイトも訪問者へ伝わります。公開する名前ごとに、それをカバーする証明書とブロックを用意し、既定ブロックでは無関係なサイトの証明書を渡すのではなく、ハンドシェイクを拒否します。

**支払いやその他の要求が二度実行される。** TLS終端が0-RTTのearly dataを受け入れ、POSTがその中で届き、攻撃者によって、またはネットワーク通信の再試行によって、同じバイト列が新しい接続で再送されます。TLS 1.3はearly dataを許可しますが、接続をまたぐリプレイからの保護は保証できません。early dataを無効にするか、ハンドシェイク完了まで処理を待つか、RFC 8470の規定に従

い、安全に再実行できない要求を425 Too Earlyで拒否します。ゲートウェイはEarly-Dataヘッダーも転送し、オリジンが理解できることを把握する必要があります。後段で待っても、前段で生じたリプレイの危険は消えません。

## 参考資料


- RFC 9846：Transport Layer Security（TLS）プロトコル、バージョン1.3 — プロトコルの版番号1.3を変えずにRFC 8446を置き換えた2026年7月の仕様です。ハンドシェイク、レコード、early dataの制限を扱います。
- RFC 9325：TLSとDTLSを安全に使うための推奨事項 — BCP 195の運用推奨事項。以下の更新と合わせて読みます。
- RFC 9852：TLSを使う新規プロトコルにはTLS 1.3が必須 — 新規プロトコルに必要な1.3のサポートと既定使用、任意の1.2相互運用を説明しています。この更新はDTLSには適用されません。
- RFC 10015：TLS 1.2とDTLS 1.2の古い鍵交換方式の廃止 — 禁止されるRSAと有限体DHの鍵交換を、証明書の署名やTLS 1.3の鍵確立と区別して説明しています。
- RFC 8996：TLS 1.0とTLS 1.1の廃止 — 二つの旧版を正式に廃止する規定と、その理由です。
- RFC 8470：HTTPにおけるearly dataの利用 — Early-Dataヘッダー、ゲートウェイに必要な連携、425ステータスを扱います。


用語集


実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

**料金を見る** お問い合わせ →

# 用語を実務に結び付ける

「TLSとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/tls/