1.1 SSLからTLSへ

デファクト標準SSL

SSLは当時急速に普及しはじめていたWebブラウザの中心的ベンダーであったネットスケープコミュニケーションズ社によって開発され同社のブラウザーに実装されました。初期のバージョンには重要な脆弱性が発見されましたが、1996年に提供されたSSL3.0は安全面でも当時としては大幅に改善されインターネット黎明期のデファクト標準としてその後広く利用されるようになりました。

中立的標準TLS

一方、インターネットの急激な普及に伴って特定企業によらないプロトコル標準の必要性に対する認識が高まってきていました。セキュリティの重要性が認識されるとともに標準策定組織としてIETFの基盤もかたまりつつあり、1996年にはIETFによるTLSの仕様策定が開始されました。しかし、最初のバージョンであるTLS1.0は中立的な組織による標準プロトコルとしての意味はあったもの、当時すでに広く普及していたSSL3.0と大きな違いがない一方互換性はなく広く普及するには至りませんでした。

標準の成熟

インターネットの世界ではその後も次々に新たな攻撃方法、プロトコル仕様レベルの脆弱性リスクが指摘され、TLSにもIETFでの改定努力が継続されました。2006年にはTLS1.1(RFC-4346)、2008年にはTLS1.2(RFC-5246)が発行されるまでの過程で、セキュリティプロトコル仕様に必要な要件も次第に明らかになりプロトコルの堅牢性も高まっていきました。またその中で、常に進化する攻撃手法、セキュリティ、暗号技術に柔軟に対応していくための仕組みも確立し、TLSはインターネットセキュリティの中核をになう標準プロトコルとして必須のものとなるに至りました。

新たなTLSの検討

しかしその一方、進化の過程で後方互換性を維持することによる複雑さの増大、新たな潜在的な脆弱性リスクも危惧されるようになり、2013年にはこれまでの経験と実績をふまえた集大成としての新しいメジャーバージョンアップ、TLS2.0に向けた検討が開始されました。しかし、TLSのバージョン情報は実際のネットワークを行きかうTLSレコード中に埋め込まれているために、スムーズにメジャーバージョンを上げることは極めて難しいことが検討開始ほどなく判明。検討目標はそのまま堅持するものの、バージョンの呼称としてはやむなくTLS1.3とマイナーバージョンアップの呼称とすることになりました。

そのような経緯でバージョンの呼称は控え目なものとなっていますが、最終的に2018年に標準として発行されたプロトコル仕様は当初の目標通りにこれまでの懸案を一掃するメジャーバージョンアップに匹敵する仕様改定となりました。

新しいセキュリティリスク、脅威

そんな中、今まで知られていなかったような大規模な組織的なネットワーク通信の傍受、情報収集の事実が知られたり、サーバソフトウェアのバグにより絶対に漏洩してはいけない秘密鍵の漏洩が発覚するなど、セキュリティの基盤を揺るがす重大な事件がおきたりもしました。今まで安全とされていた公開鍵暗号についても、その使用方法を見直す必要性が指摘されるようになり、完全前方秘匿性(第五章セキュリティの課題参照)のような新たな次元のセキュリティの必要性が認識されるようになりました。

新たな暗号技術

その間、新しい暗号アルゴリズムの研究も進み、安全性とともに従来よりも格段に処理効率のよいアルゴリズムが実用的に利用できるようになってきました。そうした個別のアルゴリズムはTLS1.2の中でも標準として順次取り入れられ、古い危殆化したアルゴリズムも徐々に廃止されてきましたが、後方互換性のためにリスクのあるアルゴリズムの一掃には至っていませんでした。また、プロトコルや暗号アルゴリズムに対する形式的検証や検証ツールについても現実の検証に有効なものが登場しはじめていました。

名前	攻撃手法	原因	解決
SLOTH	ハッシュ衝突	危殆化したハッシュ	アルゴリズム廃止
SWEET32	ブロック暗号衝突	危殆化した共通鍵暗号	アルゴリズム廃止
CurveSwap	ダウングレード	署名範囲	署名範囲拡大
LogJam	ダウングレード	署名範囲	署名範囲拡大
FREAK	ダウングレード	署名範囲	署名範囲拡大
POODLE	パディングオラクル	共通鍵暗号とMAC	AEAD
BEAST	パディングオラクル	共通鍵暗号とMAC	AEAD
Lucky 13	パディングオラクル	共通鍵暗号とMAC	AEAD
Lucky Microseconds	パディングオラクル	共通鍵暗号とMAC	AEAD
WeakDH	DHパラメータ	DHパラメータの自由度	サポートグループ
pen-and-paper	RSAパディング	PKCS#v1.5	PSS
ROBOT	RSAプライベート鍵	静的RSA	一時鍵DH
million-message	RSAプライベート鍵	静的RSA	一時鍵DH

```
TLS1.3のモチベーションとなった主な攻撃手法、脆弱性
```

TLS1.3

こうした背景から、2018年8月に正式発行されたTLS1.3ではこれまでの後方互換性を捨て、プロトコル仕様の大胆な整理が実現されました。もちろん、現実に日々使用される世界中のプロトコルを一気にバージョンアップすることは不可能です。実装上は旧バージョンのTLSも受け入れるようにしつつ新しいバージョンへの移行が行えるようになっています。しかし、新しいTLS1.3同士で確立した通信セッションに関しては、セキュリティリスクが最大限排除され、性能面からも改善が期待されるものとなっています。以下にTLS1.3の安全性と性能面での改善点についていくつかあげてみます。

安全性の改善

- ハンドシェークの大部分を暗号化
- 暗号化アルゴリズムを整理
- ダウングレード、圧縮など危険な機能の廃止

- 静的公開鍵を排除、一時鍵ディフィーヘルマン系のみに絞り込み
- 共通鍵暗号を認証鍵付き（AEAD：第三章 暗号アルゴリズム参照）のみに絞り込み

  これらに伴い、暗号スイートを大幅に絞り込み、暗号化スイート表記法を整理

性能の改善

- ハンドシェークの整理による往復数の削減、レイテンシーの低減
- 処理効率の高い新しい暗号アルゴリズム、楕円曲線の正式標準化

進化

SSLとTLSの両者はクライアントとサーバーで行われるハンドシェークを指す言葉としては今だ同義語として使われ続けています。しかし、プロトコル仕様を指す言葉としてのSSLは今や過去の仕様であり実際の製品では使われることはありません。TLSはさらにその後も進化を継続しTLS1.3で一応の集大成を実現しました。本書では、そうした認識に基づき特別の断りがないかぎりTLS1.3にもとづいて解説します。

TLS1.3はプロトコル仕様として完成度の高い標準となっているといえるでしょう。しかし、システム全体として安全なネットワーク通信を実現するためには、安全なプロトコルを実現するライブラリー層での高い品質の実現、それを使用するアプリケーション側でのセキュリティとプロトコルに関する正しい理解や使用方法、またそのアプリケーション、システムの運用方法などすべてのレイヤーで確実なセキュリティを実現する必要があります。

アプリケーションエンジニアがTLSライブラリーを正しく理解して安全なシステムを実現するために必要となる知識は多岐にわたり、それぞれの専門分野、テクノロジーのレイヤーなどに分散しがちです。本書ではそうした知識を一貫した形で理解できるように整理し、説明していくことを目指しています。

Year	Version	IETF RFC
1996	SSLv3.0	---
1999	TLS1.0	RFC2246
2006	TLS1.1	RFC4346
2008	TLS1.2	RFC5246
2018	TLS1.3	RFC8446

```
    SSL，TLSの経緯
```

1.2 簡単なTLSプログラム

1) TCPクライアント、サーバ

TLSのプログラムとプロトコルがどのように実現されているのか、簡単な例でみていきます。

TLSプロトコルはすべてTCPプロトコルによる接続の上に実現されます。図1-1に、まずTCPだけのネットワーク通信のためのクライアントとサーバの簡単なプログラムの概略を説明します。プログラム上の前処理などを省略すると、TCP通信ではまずサーバ側ではこのサーバと通信したい相手（クライアント）からの接続要求を受け付けられるよう

に待ち状態に入ります。これには、例えばBSDソケットによるプログラムではacceptを呼び出します。一方クライアントは通信したい相手のサーバに対して接続要求を出します。BSDソケットではconnectを呼び出します。この要求がサーバに受け入れられるとTCP接続が成立し(図1-1①)、クライアントとサーバの間でTCP通信ができるようになります。

その後、この接続を使ってクライアントとサーバの間でアプリケーションの必要に応じたメッセージの送信、受信を繰り返します(図1-1②)。

必要なメッセージの送受信が完了したらTCP接続を切断します１(図1-1③)。

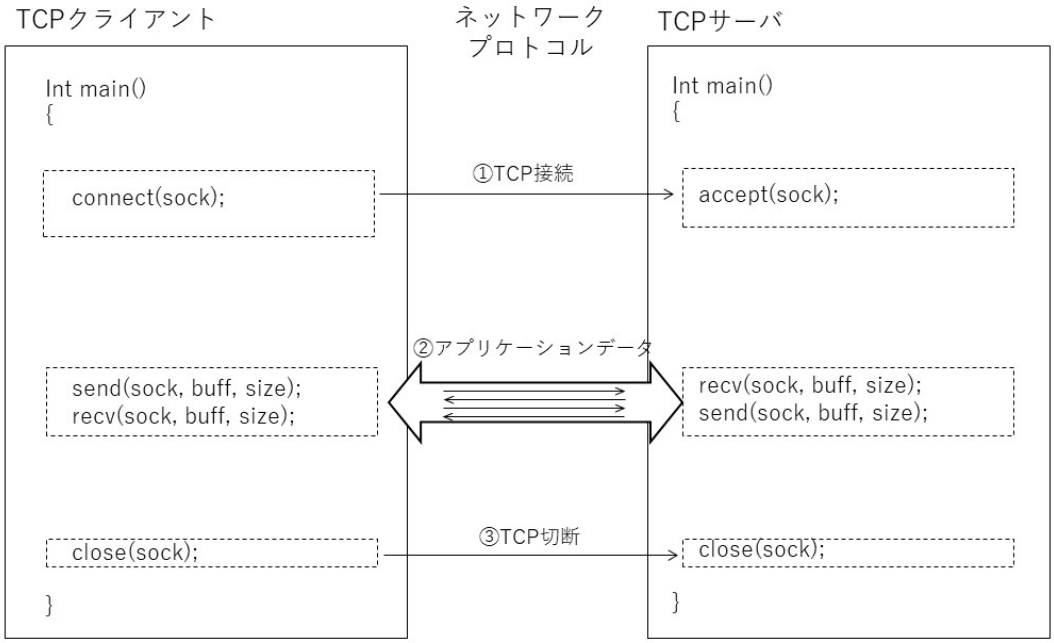

図1-1: TCPログラムとプロトコル

2) TLS層を追加する

図1-2はこれに対してTLSのプログラムを追加したものです。TLSは全ての通信をTCPプロトコルの上でおこなうのでTCPプログラムの接続（図1-2①)、切断処理（図1-2⑤)は図1-1とまったく変わりません。TLSのすべてのレコードはTCP接続されたクライアント、サーバ間のTCPレコードの上に乗せて転送することになります。

次に、サーバ側プログラムではTLSレイヤーの接続要求を待つためにSSL_accept、クライアント側プログラムでは接続要求のためにSSL_connectを呼び出します。この呼び出しで一連のTLSハンドシェークが実行されTLS接続が確立します（図2-1③)。

TLS接続が確立したら、目的とするアプリケーションデータの送受信を行います（図2-1④)。これはプログラム上ではSSL_send/SSL_recv APIによって行います。アプリケーションが送信したい平文のメッセージはSSL_sendによって暗号化され、SSL_recvによって復号化され相手方のアプリケーションに平文で引き渡されます。この時、受けとったメッセージが送信元メッセージから改ざんされていないこと、真正性のチェックも行います。

アプリケーションデータの送受信が完了したらTLS接続、TCP接続の順に切断します（図2-1⑤、⑥)。

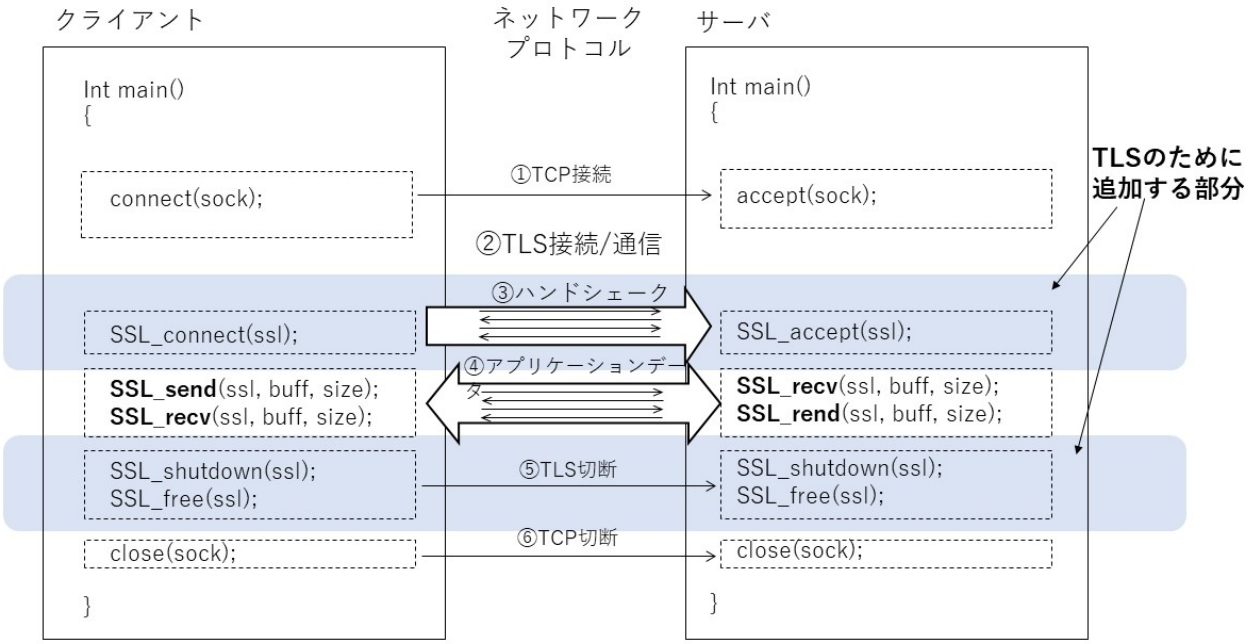

図1-2: TLSプログラムとプロトコル

TLSプロトコルを見てみる