**## はじめに**

第一章では、TLSプログラミングのベースとなっている技術について見ていきます。第二章ではTLSプロトコルについて、第三章ではそこに使われている暗号アルゴリズム、技術について、また、第四章では、関連する各種の標準について解説します。また、第五章ではTLSプログラミングにおいてセキュリティ上考えておくべき事項についてまとめます。

**## 2.1 簡単なクライアント⇔サーバー通信**

図2-1は、TLSではじめて通信するクライアントとサーバのプログラムと、両者の間のネットワークプロトコル（フルハンドシェーク）の概要を示しています。プロトコルとしてのTLSはTCPの上に実現されています。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。

これを実現するために、クライアントとサーバの間ではまずTCP接続を確立させます。例えばBSDソケットによるプログラムでは、サーバ側はクライアントからの接続要求待ち(accept)に入り、クライアントから目的のサーバに対して接続(connect)を要求することで実現します。

以後のTLSのメッセージはすべてこの接続によるTCPレコードに乗せて運ばれます。

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。

**## 2.2 フルハンドシェイク**

**### 2.2.1 フルハンドシェイクの目的**

2.1のようにクライアントがはじめてサーバとTLS通信する際は、サーバは通信相手のクライアントに対して事前情報なしに安全なTLSセッションを確立する必要があります。これを行うのがフルハンドシェークです。

フルハンドシェークの主な目的は次の３つです。

１）通信の両者で使用する暗号スイートを合意すること<br>

２）セッションで使用する一連の鍵を合意すること（鍵合意）<br>

３）通信相手が正しい相手であること、成りすましがないことの確認すること（ピア認証）<br>

ピア認証とは、クライアントがサーバの正当性を認証するサーバ認証と、サーバがクライアントの正当性を認証するクライアント認証の二つです。TLSの場合、サーバ認証は必須、クライアント認証はオプションとなっています。

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。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。

**### 2.2.2 暗号スイートと鍵合意**

ハンドシェークの主要な目的である暗号スイートと鍵の合意は、クライアントからサーバに最初に送られるClientHelloレコードとそれに対するサーバからの応答ServerHelloによって行われます。図2-2はこの様子を表したものです。

<br>

ClientHelloレコードの各TLS拡張には以下のような情報が格納されます。

１）サポートするTLSバージョンのリスト<br>

２）サポートする暗号スイートのリスト<br>

３）サポートする楕円曲線暗号の曲線リスト<br>

また、Key Share拡張にはディフィーヘルマン（楕円曲線DHを含む）鍵合意のためのDHパラメータとクライアント側のDH公開鍵を格納します。Key Shareは複数の候補を含むことができます。

これに対してサーバ側は、提示されたリストに合意できるものがあれば各項目の合意する内容をServerHelloにて返却します。この時、ServerHelloのKey Shareにはサーバ側のDH公開鍵も格納します。それらを受け取った両者はDHアルゴリズムによってプレマスターシークレットを算出します。この値をもとにHKDF鍵導出アルゴリズムによってその後の共通鍵暗号による暗号化、復号化のための鍵、IVを導出します。TLS1.3では鍵、IVは送信元（サーバ側、クライアント側）によって、またハンドシェーク中、アプリケーションデータ転送、0-RTTなどの用途により異なる値を導出しセッション鍵とし安全性をさらに高めています。

クライアント側が提示したKey Shareのリストに対してサーバ側が合意できない場合はクライアントに対して１回だけ別の候補を要求することができます(HelloRetryRequest)。これに対して、クライアント側は２度目のClientHelloで次の候補を示します。サーバ側がこれに合意できれば、ハンドシェークは次に進みます。合意できない場合は、ハンドシェークは中断します。

なお、TLS1.3ではこのようにClientHelloとServerHelloレコードの1往復のやりとりでセッション鍵の導出まで完了しますが、TLS1.2以前では合意のためのサーバ鍵、クライアント鍵は次のServerKeyExchange, ClientKeyExchangeに交換されるため合意の完了のために２往復を必要としていました。また、TLS1.3ではハンドシェークの最初でセッション鍵が導出できるので、それ以降のハンドシェークを秘匿することが可能となり安全性を高めています。

**### 1) TLSバージョンの合意**

ネットワーク上で複数のTLSバージョンのプロトコルが混在できるように、ClientHelloではサポートする複数のバージョンを提示することができるようになっています。これに対してサーバ側は合意したバージョンに対応する形式のServerHelloを返送します。これによって、それ以後のハンドシェークは合意したバージョンの形式で進めることができます。TLS1.3では、Secure Renegotiationは廃止されたので、ここでTLS1.3を合意した場合はその後すべて1.3に準拠する必要があります。

また、TLS1.3ではダウングレードは認められないので、クライアントが1.3を含む複数のバージョンを提示した場合でも、サーバ側はもしTLS1.3をサポートするならば必ず1.3で合意しTLS1.3のServerHelloを返却する必要があります。この時、ClientHelloに示される暗号スイートリストにTLS1.3スイートが無い場合はダウングレードの一種とみなされハンドシェークは終了します。また、TLS1.3で合意することを期待するクライアント側はこの値が規定されている値ではないことを確認し、ダウングレード攻撃を防止します。<br>

一方、サーバがTLS1.3を含む複数のバージョンをサポートする場合に、クライアント側がTLS1.2以下のみサポートする場合は、サーバ側もTLS1.2以下しかサポートしない場合と等価の動作をすることが認められています。ただし、その場合にはサーバランダムの後尾にその旨を示す特定のバイト列を表示します。<br>

表2-1にクライアント、サーバでサポートするTLSバージョンの組合わせと求められる動作についてまとめます。

|クライアント<br>TLS1.2以下|<br>TLS1.3|サーバ<br>TLS1.2以下|<br>TLS1.3|動作|

|---|---|---|---|---|

| |✓| |✓|TLS1.3セッション|

|　|✓|✓|✓|TLS1.3セッション|

| |✓|✓| |ハンドシェーク終了|

|✓|✓| |✓|TLS1.3セッション|

|✓|✓|✓|✓|TLS1.3セッション|

|✓|✓|✓| |TLS1.2以下セッション|

|✓|　| |✓|ハンドシェーク終了|

|✓| |✓|✓|TLS1.2以下セッション<br>注|

|✓| |✓| |TLS1.2以下セッション|

表2-1 TLSバージョンの合意

注：サーバランダムの後尾８バイトに<br>

TLS1.2の場合十六進"44 4F 57 4E 47 52 44 01"<br>

TLS1.1以下の場合十六進"44 4F 57 4E 47 52 44 00"<br><br>

**### 2) 暗号スイート**

TLS1.3ではそれまでのバージョンにくらべて以下のように利用可能な暗号スイートの種類が大幅に整理されました。

１）鍵合意アルゴリズムとしては静的RSAを廃止し、一時鍵DH（楕円曲線DHを含む）のみとなりました。<br>

２）静的RSAが廃止されたため、証明書はピア認証のみに使用することになりました。これによって、鍵合意と認証は完全に分離、独立した扱いが可能となりました<br>

３）危殆化した共通鍵暗号アルゴリズムが大幅に整理されました<br>

４）MACによる真正性検証を廃止し、認証付き暗号(AEAD: Authenticated Encryption with Associated Data)アルゴリズムのみに整理されました<br>

５）ハッシュはHKDF鍵導出のためのハッシュアルゴリズムのみを指定するものとしました<br>

１）の整理により暗号スイート表記上は鍵合意アルゴリズムの表記は意味がなくなり、TLS1.3向け表記からは削除されました。また２）により、証明書に関する情報も暗号スイートとは分離し、必要な情報はTLS拡張に格納しています。

以上の結果、TLS1.2までに数百にのぼっていた暗号スイートは、現在TLS1.3として利用できるものとして以下の通りに絞り込まれています。

<br>

|名前|ID|

|:---:|:---:|

|TLS\_AES\_128\_GCM\_SHA256|0x1301|

|TLS\_AES\_256\_GCM\_SHA384|0x1302|

|TLS\_CHACHA20\_POLY1305\_SHA256|0x1303|

|TLS\_AES\_128\_CCM\_SHA256|0x1304|

|TLS\_AES\_128\_CCM\_8\_SHA256|0x1305|

表2-2 TLS1.3の暗号スイート

<br>

鍵合意に関する情報は暗号スイートから分離され、ECDHに使用できる楕円曲線の種類については、別途TLS拡張(Supported Group)にリストを示します。Groupの種類として標準的な楕円曲線の種別ID、またDH向けには鍵長を示すIDが定義されています。

表2-3にSupported Groupで使用される主なGroupとIDの一覧を示します。

**### 3) 暗号スイートの合意**

ClientHello, ServerHelloの往復では、これらの暗号スイートとともにSupported Groupで示される楕円曲線の種類を合意します。この時、TLSバージョンとしてTLS1.3を合意している場合、必ずTLS1.3の暗号スイートとKey Shareに示されるDHパラメータを合意する必要があります。

クライアントから示されるものに合意できるものが無い場合、サーバは再度のClientHello要求 (HelloRetryRequest) を一回だけ発行することができます。

**### 4) 鍵合意**

TLSバージョンと暗号スイートに合意したら、サーバはKey Shareに示されるDHパラメータとクライアントのDH公開鍵を受け取ります。また、これに対してServerHelloでサーバのDH公開鍵を返します。これにより、両者それぞれでPre-Master Secret、またHKDFによりセッション鍵を導出します。

**## 2.1.3 ピア認証**

ハンドシェークのもう一つの主要な目的はピア認証（クライアントによるサーバ認証、サーバによるクライアント認証）です。TLSではサーバ認証は必須、クライアント認証はオプショナルです。しかし、サーバ側がクライアント認証を要求した場合はクライアント側は必ずそれに応答する必要があります。

図2-3は、クライアントとサーバプログラムとサーバ認証に使用される証明書や鍵、またプロトコルとの関係を示します。TLSプログラム上、クライアント側ではサーバ認証のためにあらかじめに信頼するCAの証明書をロードしておきます。サーバ側では、CAによって署名されたサーバ証明書とプライベート鍵をロードしておきます。

ハンドシェークでは、サーバ側はロードされたサーバ証明書をCertificateレコードにてクライアントに送信します。また、プライベート鍵によって作成した署名をVerifyCertificateレコードにて送信します。受け取ったクライアント側では、ロードされたCA証明書で送られてきた証明書の真正性を検証したうえで、格納されている公開鍵によって署名を検証します。

クライアント認証では、ほぼこれと同じことをクライアントとサーバを対称にした形で行います。ただしクライアント認証の方はオプショナルなので、サーバ側は必要に応じてクライアントに対して認証要求(CertificateRequest)を送信します。

ピア認証のプロトコルに関して、TLS1.2まではサーバ側から送られる署名はServerKeyExchangeに格納されるなど、プロトコル上一部対称でない部分がありましたが、TLS1.3では上記のように認証側、非認証側で対称となるように整理されました。

ピア認証に関するアルゴリズム情報は各々のTLS拡張に格納されています。

証明書の署名アルゴリズムとしてサポートするものは、署名アルゴリズムTLS拡張(Signature Algorithms)にそのリストが格納されます。署名アルゴリズムは署名とハッシュアルゴリズムの組み合わせで規定されます。TLS1.3では、署名部分としてRSAとECDSAが標準として定義されています。RSAの場合、パディング方式も定義します。ECDSAの場合、楕円曲線の種類もここで規定します。ハッシュとしてはSHA1またはSHA2が使用されます。

表2-4にTLS1.3で使用できる署名アルゴリズムの一覧を示します。

**## 2.2.3 Hello Retry**

**## 2.3 事前共有鍵とセッション再開**

**### 事前共有鍵**

事前共有鍵(PSK: Pre-shared Key)を利用すると、通信する両者が別途何らかの方法で鍵を合意しておいてそれを利用して通信することができます。

**### 2) 鍵交換付き事前共有鍵**

**### 3) セッション再開**

TLS1.3では、セッション再開は前に確立したTLSセッションにおいて交換しておいたセッションチケットを鍵として使用するPSKの一種として整理されています。

図2-5にPSK

**### 4) Early Data**

**## 2.4 その他の機能**

**### 2.4.1 その他のTLS拡張**

**#### 1) SNI**

**#### 2) ALPN**

**### 2.4.2 ハンドシェーク後のプロトコル**

**#### 1) 再鍵生成**

**#### 2) 再認証**