TLS: Transport Layer Security

(a.k.a.: Secure Sockets Layer (SSL))

Network Security
総合情報学科

TLS (SSL)とは?

- 二者間を結ぶ安全な通信路を提供するプロトコル
- 目的: Online Transaction(電子商取引)に信頼を提供
  - 「Webによる電子商取引」の増加が契機
- オンライン取引で望まれること ⇒ 「信頼」
  - 「安全・確実」に情報を伝達

    ⇒ 個人情報やCredit Card情報の伝達
  - 意図する通信相手か？

    ⇒ 通信相手が意図する相手(取引会社)か？

提供機能

２点間の「安全」な通信路の確立

- 通信内容が第三者に漏えいしない
- 通信内容が改変されない
- 通信相手の検証

上記に加えて「透過性」

提供機能 (Cont.)

- 通信内容が第三者に漏えいしない
  ⇒ 機密性 (Confidentiality)

- 通信内容が改変されない
  ⇒ メッセージ完全性 (Message integrity)

- 通信相手の検証
  ⇒ エンドポイント真正性 (Endpoint authentication)

透過性

- 当初：Webによるオンライン取引が対象

  ⇒ HTTP (=HyperText Transfer Protocol)を安全にする

- 広範なSecurity問題の解決に乗り出す

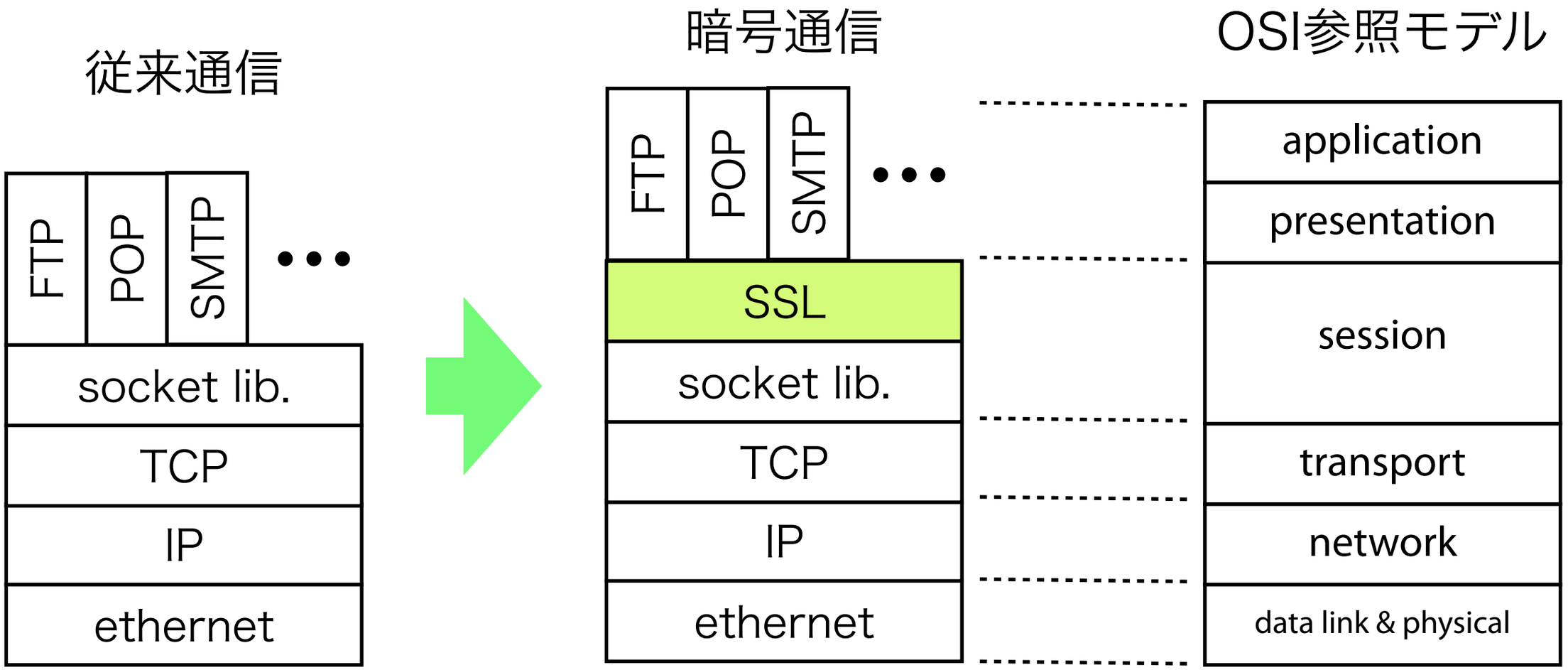
  - 対象をHTTP通信からTCP通信に変更

  - TCPによるApplication通信ならばTLSを適用可能に

    ⇒ **「透過性」**

TLSが成功を収めた最大の要因

透過性



- OSI参照モデルのSession層に該当 (transport層または両層の境界で動作という見解もある)

  TLSはLayerの1つとして機能
  上位層Protocolには非依存 (=透過性)
  ⇒ 種々のアプリケーション通信の保護が可能 (要Code修正)

API levelでの比較

Client program in TCP

```
int socket(int, int, int)
int connect(int, const struct sockaddr *, int)
ssize_t write(int, const void *, size_t)
ssize_t read(int, void *, size_t)
```

Client program with SSL

```
SSL * SSL_new(SSL_CTX *)
int SSL_connect(SSL *)
int SSL_write(SSL *, void *, int)
int SSL_read(SSL *, void *, int)
```

Transport Layer Security(TLS)とSSL

- Secure Sockets Layer(SSL)は、一民間企業により開発
  - Netscape Communications社
- 1990年代、Webブラウザ最有力企業
  - Webブラウザ：“Netscape Navigator”
- “SSL”はNetscapeの社内技術者により設計
  - 社外からの意見が反映される余地はなかった

Transport Layer Security(TLS)とSSL

- 1996年 : SSLと同等機能を持つProtocolの標準化開始
  - IETF / **TLS**ワーキンググループ
  - 以下の問題にはばまれ、仕様策定が遅延したが...
    - 標準サポートの暗号化手法の選択
    - 暗号技術の輸出規制
    - PKIXの作業遅延の影響
- 1999年1月：RFC 2246 (TLS Protocol ver 1.0)として公表

PKIX: IETF public key infrastructure working group

SSL/TLSの歴史

version	策定年度	策定者	
SSL 1.0	1994	Netscape	設計reviewの段階で脆弱性発覚。破棄。 実装製品なし
SSL 2.0	1994		ダウングレード攻撃の脆弱性が発見。 現在、主要Webブラウザでは無効化
SSL 3.0	1995		version 2.0の問題修正。機能追加 致命的な脆弱性発覚、無効化
TLS 1.0	1999	IETF	SSL 3.0とほぼ同等機能
TLS 1.1	2006		AES暗号を追加
TLS 1.2	2008		SHA256を追加
TLS 1.3	?		

TLSとSSLの関係

- 呼称：
  - どちらの技術も “SSL” と呼ぶことが多かった
- Version番号
  - TLS 1.0, 1.1, 1.2 が正式名称
    が、session確立時のversion No.は3.1, 3.2, 3.3
- 下位互換性
  - RFC 6176 (Mar., 2011)で下位互換性を排除、特に
    SSL v3.0以下でセッション確立しないように修正

Version Num.
1.0
2.0
3.0
3.1
3.2
3.3

脆弱性が存在するため

下位互換性

- Session確立時にClient, Server間で調整(negotiation)
- ClientとServerの双方で対応可能なversionを提示
  ⇒ 両者が対応可能な最新のversionを選択

最新 versionは，　下位(過去)の
全versionの機能も持つ.

という互換性 ではない

TLS/SSLの適用方法

- ポート番号による分離
- URL転送 (HTTP Redirect)
- アプリケーションプロトコルによる調整

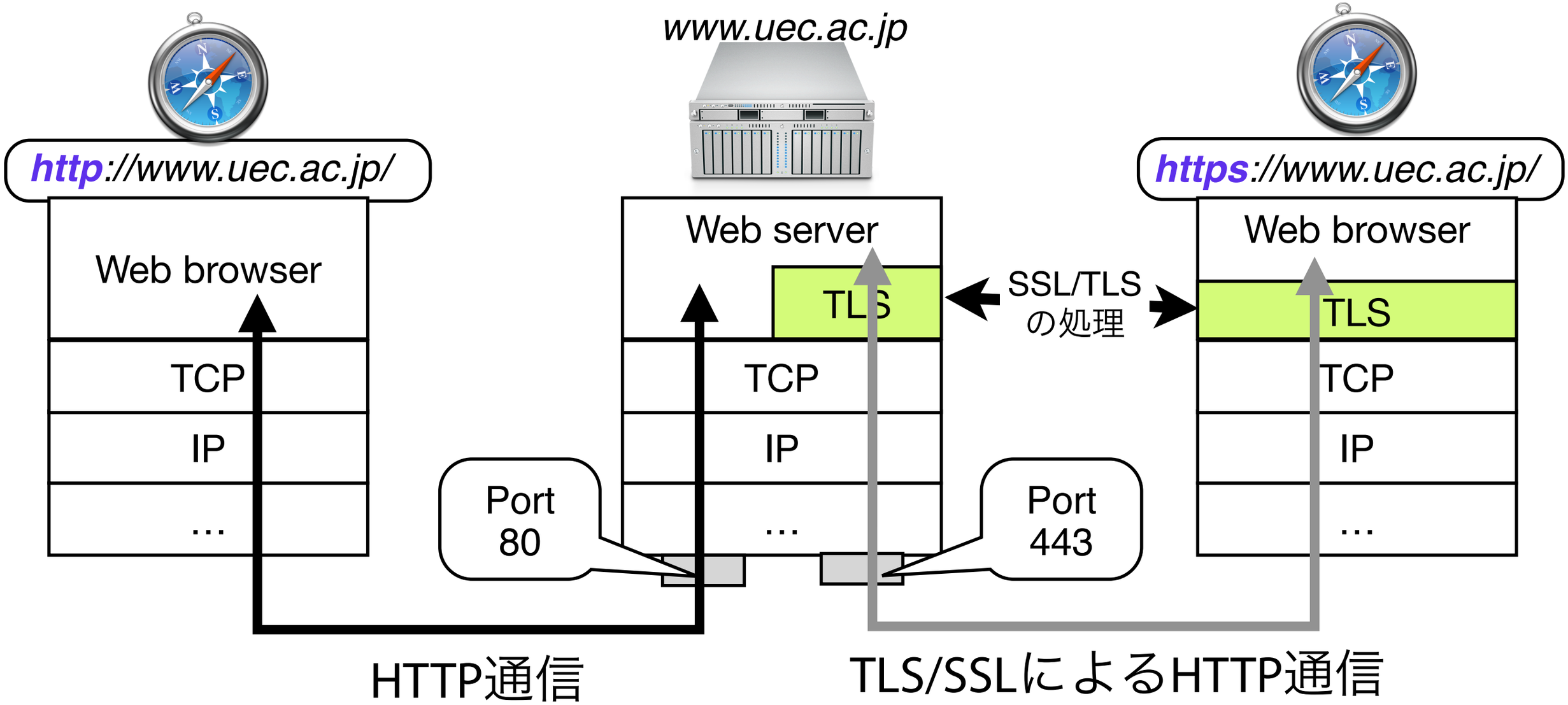
  Upward negotiation (STARTTLS)

ポート番号による分離

- 1つのアプリケーションProtocol毎に「TLSあり」と「TLSなし」の通信路をサーバ側で別々に用意
  - ポート番号で区別
    - Webの場合： **HTTP**(Port 80)と**HTTPS**(Port 443)
    - Web以外のApp. Protocolも同様
      - 例) Service Name and Transport Protocol Port Number Registry (Web)で “over TLS/SSL” で検索
- 「TLSあり」ポート
  ⇒ 通信開始前に必ずTLSによる接続確立処理を実施

App. Protocol = Application Protocol

ポート番号による分離


Web server:
www.uec.ac.jp
http://www.uec.ac.jp/
https://www.uec.ac.jp/
Web browser
TCP
IP
…
Web server
TLS
TCP
IP
…
SSL/TLS
の処理
Web browser
TLS
TCP
IP
…
Port
80
Port
443
HTTP通信
TLS/SSLによるHTTP通信

URL転送

- とあるURLへのアクセスを異なるURLに自動的に転送
  - 「HTTP(TLSなし) URL」へのWebアクセスを
    「HTTPS (TLSあり) URL」へ**転送 (redirect)**
    - Web(HTTP)限定だが、手軽に実装可
- 実装方法
  - HTTP：HTTP Redirect
  - HTML：(http-equiv=”refresh”) 命令の<meta>タグ
  - JavaScript ：“location.href =“ コード

Application protocolによる対応

- Node AからNode Bに対し「TLS通信希望」という Messageを送信、それに応答する形でTLS通信を始める
    - 1) 接続確立当初はTCP通信 で接続
    - 2) 片方の*Node*が“TLS通信”への移行表明
    - 2) 双方(Server, Client)がTLS通信への移行に合意
    - 3) TLSによる接続確立処理を開始
- 重要：ポート番号は１つでOK (App. Protocol内で切り替える)
    - 「TLSなし」通信のポート番号をTLS通信でそのまま利用

App. protocolの
機能として実装

App. proto.による対応 (Cont.)

- Upward negotiationと呼ばれる
  - 平文通信(下)から暗号化通信(上)へ 移行
- 利点
  - Client側(利用者)がTLS利用を意図しなくて ”も” よい
  - Firewallの設定変更不要 (Port番号が変わらないため)
- 欠点
  - Client, Serverともにシステムの改良が必要
  - 通信コスト増

App. Proto. = Application Protocol

STARTTLS

- 通信プロトコルの拡張によるTLS対応
  - 平文通信 ⇒ 暗号化通信へ：通信途中から「移行」
    - 仕組み自体はアプリケーション非依存
- いくつかのApp. protocolで規定されている
  - SMTP, IMAP, POP3(電子メール関連)、FTP(ファイル転送)、LDAP(ディレクトリサービス)など

利用されている要素技術

- 通信路の暗号化
  - 公開鍵暗号、共通鍵暗号
- 認証 (Server認証、Client認証)
  - 公開鍵(電子)証明書、Public Key Infrastructure (PKI): 公開鍵基盤
- 通信データの改ざん検出
  - ハッシュ関数、電子署名