TLS 1.3とその周辺の標準化動向

奥 一穂

2018年4月

自己紹介

- CDN企業「Fastly」のプログラマ
- HTTP/2実装「H2O」の主開発者
  - picotls (TLS 1.3), quicly (QUIC) も
- 最近初めてのRFCが出ました
  - RFC 8297 – Early Hints for HTTP

Agenda

- TLS 1.3
  - Using Early Data in HTTP
  - Ossification
- DTLS 1.3
- Exported Authenticators
  - Secondary Certificates for HTTP/2
- Certificate Compression
- SNI Encryption

TLS 1.3, Early Data, Ossification

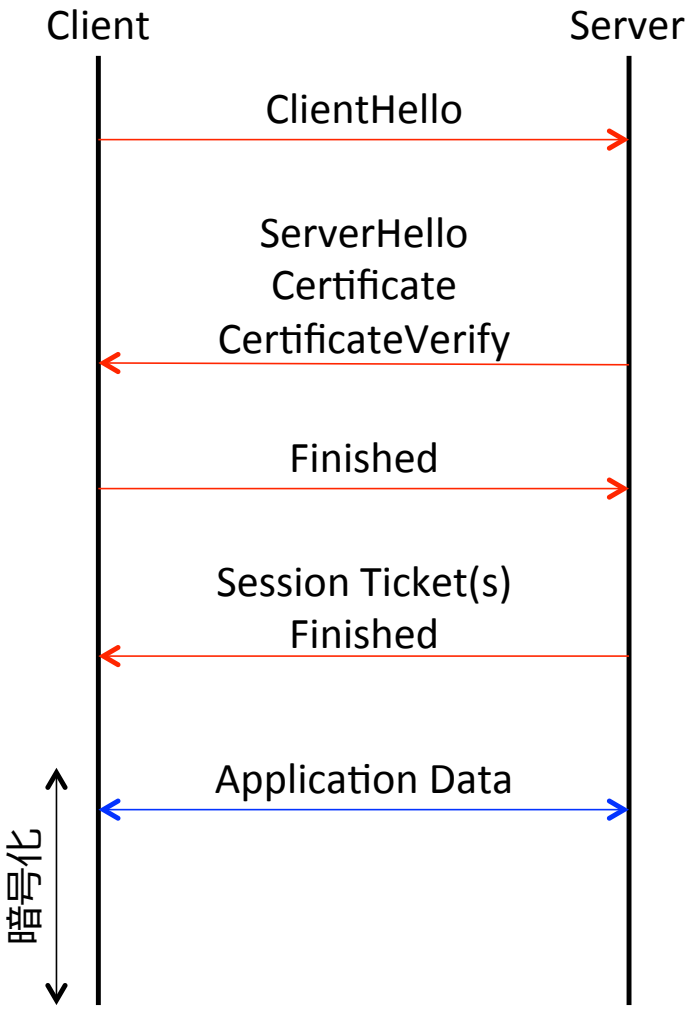
TLS 1.3

- 実質 TLS 2.0
- draft-28
- Submitted to IESG for Publication

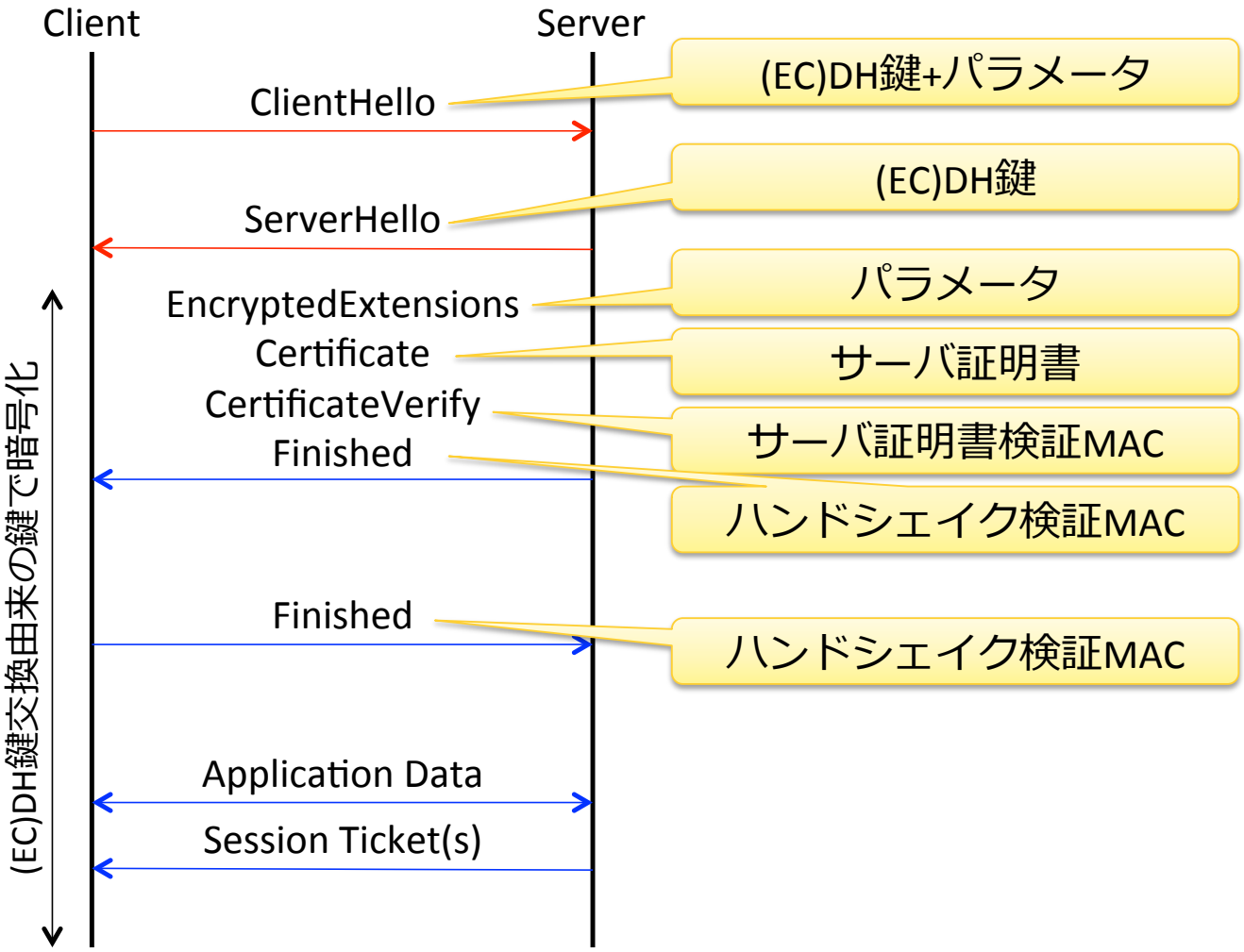
TLS 1.3の特徴

- ハンドシェイクの再設計
  - 0〜1 RTTでの接続確立
  - 暗号化
  - Forward Secrecy (前方秘匿性)
- トラッキング抑止
  - Pervasive Monitoring is an Attack (BCP188, 2014)
  - ワンオフのセッションチケット
  - 証明書の暗号化
- AEAD前提のレコードレイヤ

TLS 1.2の通信フロー



TLS 1.3の通信フロー



ハンドシェイクの考え方

- TLS 1.2
  - パラメータ交換の後に公開鍵・証明書を交換
- TLS 1.3
  - いきなり公開鍵交換
    - 同時にパラメータ交換
    - 公開鍵の方式が異なる場合はリトライ
  - 鍵交換が終わったら暗号化
    - その後に証明書交換

HelloRetryRequest

- 特殊なServerHello
  - randomフィールドのマジックナンバーで識別
    - CF 21 AD 74 E5 9A 61 11 BE 1D 8C 02 1E 65…
  - ClientHelloの再送信を要求

パラメータ交換

- ClientHello / ServerHello
  - 平文のパラメータ
  - クライアントが最初に送信、サーバが応答
- EncryptedExtensions
  - 暗号化されたパラメータ
  - サーバが送信

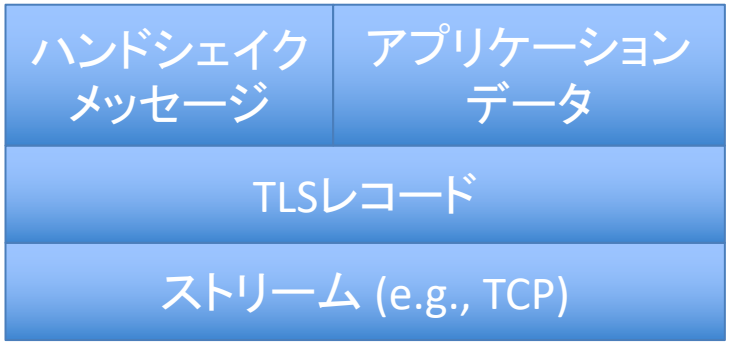
Extension	TLS 1.3
server_name [RFC6066]	CH, EE
max_fragment_length [RFC6066]	CH, EE
status_request [RFC6066]	CH, CR, CT
supported_groups [RFC7919]	CH, EE
signature_algorithms [RFC5246]	CH, CR
use_srtp [RFC5764]	CH, EE
heartbeat [RFC6520]	CH, EE
application_layer_protocol_negotiation [RFC7301]	CH, EE
signed_certificate_timestamp [RFC6962]	CH, CR, CT
client_certificate_type [RFC7250]	CH, EE
server_certificate_type [RFC7250]	CH, EE
padding [RFC7685]	CH
key_share [[this document]]	CH, SH, HRR
pre_shared_key [[this document]]	CH, SH
psk_key_exchange_modes [[this document]]	CH
early_data [[this document]]	CH, EE, NST
cookie [[this document]]	CH, HRR
supported_versions [[this document]]	CH, SH, HRR
certificate_authorities [[this document]]	CH, CR
oid_filters [[this document]]	CR
post_handshake_auth [[this document]]	CH
signature_algorithms_cert [[this document]]	CH, CR

セッション再開

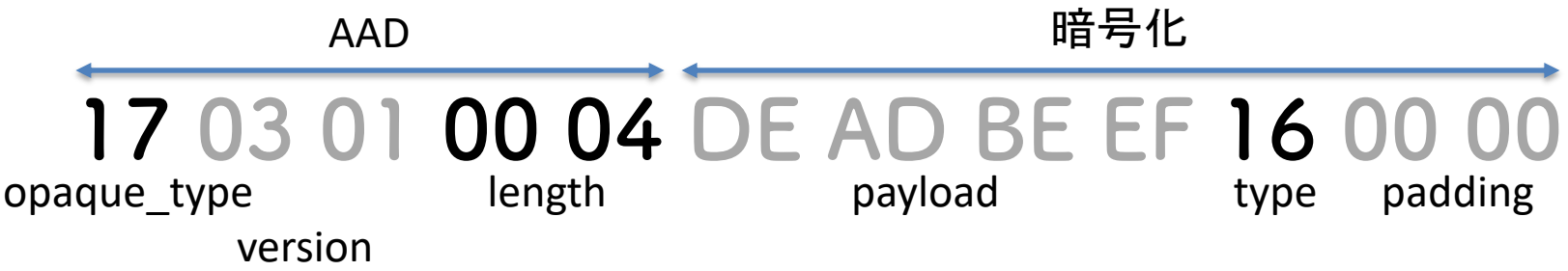
- TLS 1.2
  - HelloでIDを交換
    - ステートフル（サーバ側で記憶する必要）
  - Session Ticket Extension (RFC 5077)
    - ステートレス（暗号化されたクッキーを配布）
  - どちらもハンドシェイク中に平文で送信
    - TCP接続をまたぐユーザトラッキングが可能
- TLS 1.3
  - ハンドシェイク完了後にticketを配布
  - ticket使用は1回のみ

レコード

- TLSで暗号化する単位
- 例: 16 03 01 00 04 DE AD BE EF
  - type: 16
  - version: 03 01
  - length: 00 04
  - payload: DE AD BE EF
- type
  - 15: alert
  - 16: handshake
  - 17: application_data

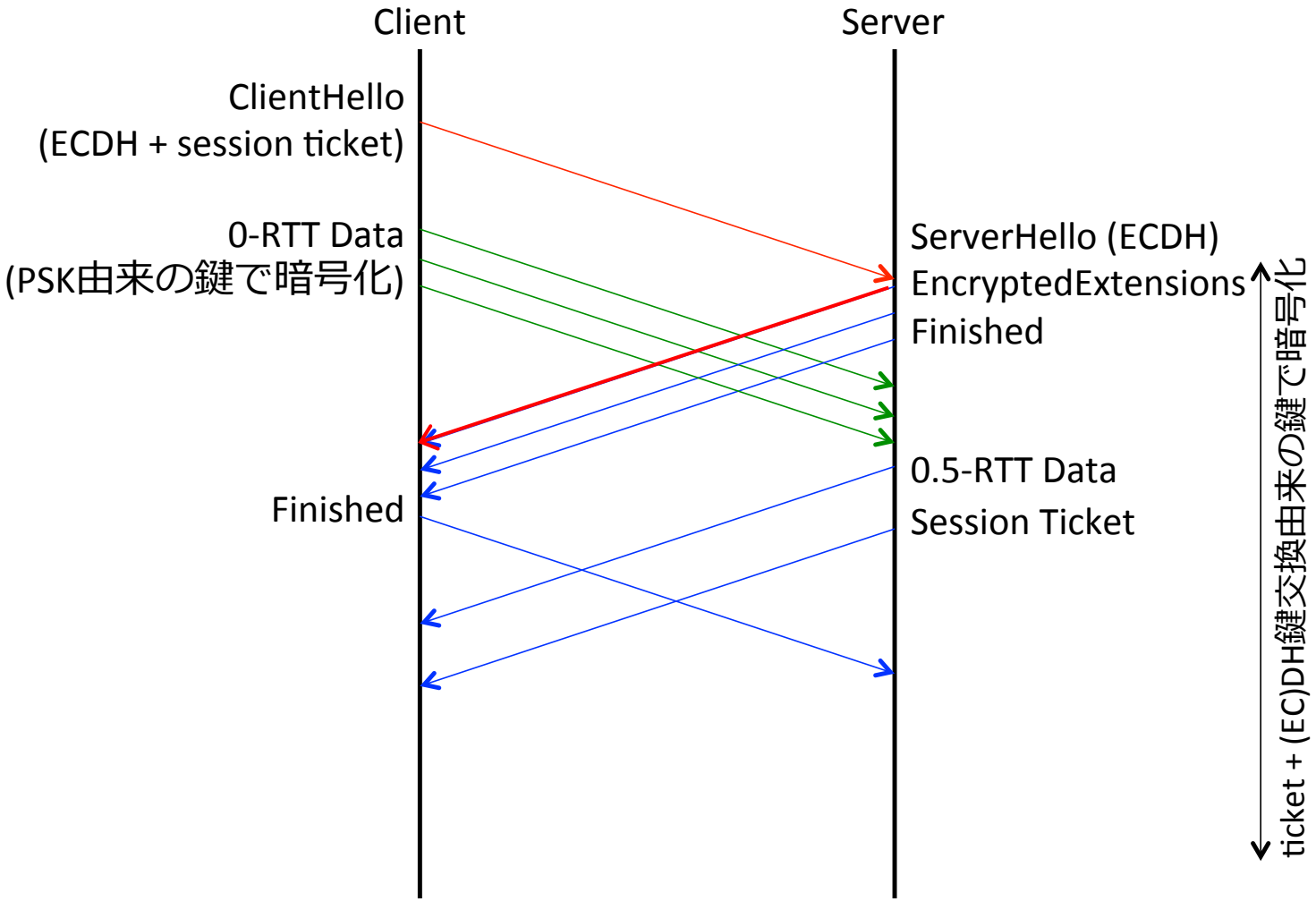


レコード



- 17はAEAD暗号化を表すtypeに変更
- 本当のtypeは暗号文の中に
- パディングは任意個数のゼロ
- AEAD暗号 = Authenticated Encryption with Additional Data

TLS 1.3の通信フロー（0-RTTリザンプション）



0-RTT Data

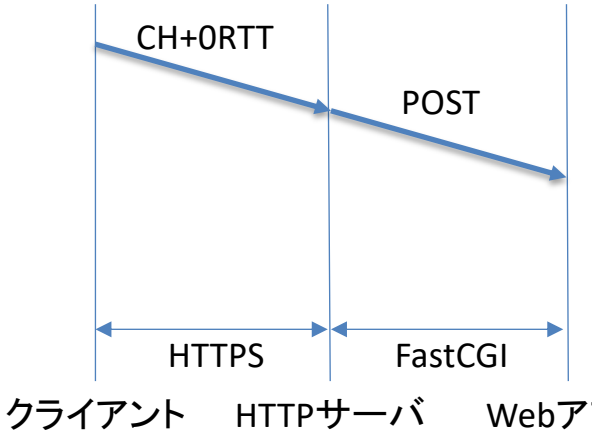
- PSK由来の鍵で暗号化
- 終端はEndOfEarlyDataハンドシェイクメッセージで伝達
- サーバは0-RTTを解読できないとき、どうする?
  - trial decryption!
  - 0-RTTが解読できなくても、EOEDはハンドシェイクメッセージなので解読可能

0-RTTの問題

- リプレイ可能
  - 攻撃例: 銀行振込要求をコピーしてリプレイ
- 対策:
  - リプレイ可能な時間幅を限定
    - チケットの(難読化された)ageを利用
  - サーバ側のbloom filterで検出
    - サーバが複数拠点に分かれていたら?
  - アプリケーションプロトコルで判定
    - リトライ安全じゃない情報の処理は、ハンドシェイク完了まで遅延

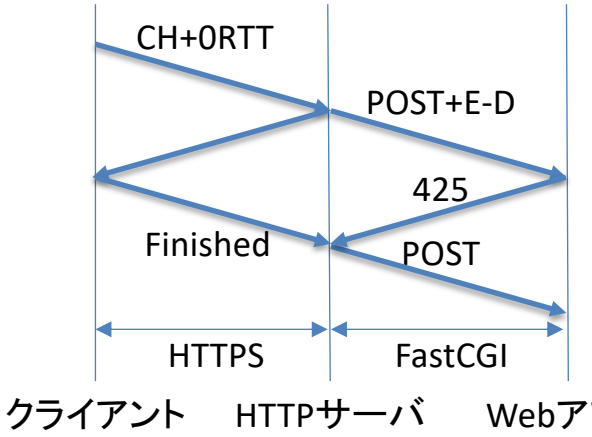
HTTP vs. 0-RTT

- べき等性があるリクエストは問題ない
  - 例. 画像のGET
- べき等性の有無はWebサーバでは判定不能
- Webアプリに、0-RTTリクエストであることを伝え、判定させる仕組みが必要



Using Early Data in HTTP

- Finished以前に受信したリクエストを転送する場合は、Early-Dataヘッダをつける
- サーバの挙動:
  - 0-RTTもしくはE-Dつきリクエストについては、425 Too Earlyを返しても良い
- クライアントの挙動:
  - 425を受信したらFinished送信後にリクエスト再発行



Using Early Data in HTTP

- 中継者の挙動
  - 0-RTTリクエスト転送時はE-D付加
    - E-Dつきのリクエストはそのまま転送
  - 425を受信したら
    - 自分がE-Dつけた場合は、Finishedを待って再発行しても良い
    - それ以外はクライアントに425送信

Ossification

- 最終局面でエラーレートが問題に

	TLS 1.2	TLS 1.3
Chrome (-18)	1.7%	7.7%
Firefox (-23?)	2.2%	3.9%

https://datatracker.ietf.org/meeting/100/materials/slides-100-tls-sessa-tls13-02

- 原因: ユーザ側のファイアウォール
  - 例: サーバ証明書を見て検閲する企業向製品
    - 1.2までは可能, 1.3では証明書が暗号化されている

「互換モード」の策定

- TLS 1.2のセッション再開っぽく見せる
  - セッション再開では証明書転送しないため
- 偽のSession IDをHelloに入れる
- 1.3では使われなくなったCCSメッセージを、1.2と同じタイミングで送信
- HelloRetryRequestはServerHelloっぽく
  - randomフィールドのマジックナンバーで判定

他のOssification事例

- MPTCP策定時の諸問題
- TCP FastOpenのエラーレート
- 「TCP最適化」装置によるパフォーマンス劣化
- TLSのレコードバージョン
- Google QUICの"07 octet"問題

QUIC vs. Ossification

- Ossifyして良いフィールドをInvariantsとして定義
  - 例: Connection ID
- それ以外のフィールドは全て暗号化、難読化、グリーシング

DTLS 1.3

Exported Authenticators

HTTPとクライアント認証

- HTTP/1
  - 一度に流れるリクエストは1つ
  - TLSのクライアント認証で十分
- HTTP/2
  - 複数のリクエストが同時に流れる
  - リクエストごとに異なるクライアント証明書を使いたい

HTTP/2とサーバ認証

- 異なるドメインへのリクエストでも既存のHTTP/2接続を使いまわしたい
  - 接続確立時間の短縮、INITCWNDに縛られない初期転送速度
- 複数のサーバ証明書をクライアントに送りたい

共通の課題

- 複数の証明書をどうやって転送するか
- 対応する秘密鍵の保有をどうやって証明するか

Exported Authenticators in TLS

- 証明書とその所有証明の(要求と)送信
  - 送受信方法は、TLS接続上で動作しているアプリケーションプロトコルにおまかせ
- 証明書の送信方法
  - TLS 1.3のハンドシェイクメッセージを再利用
    - Certificate, CertificateVerify, Finished
    - TLS接続からエクスポートした秘密情報を署名することで認証
- 証明書の要求方法
  - TLS 1.3のCertificateRequestメッセージを再利用

Secondary Certificate Auth. in HTTP/2

- HTTP/2はTCP上に複数のストリームを重畳
  - 2種類のストリーム: 制御用,リクエスト送受信用
  - ストリームは複数のフレームから構成
- 証明書とリクエストは1:n対応
- 証明書関連の情報はフレームで交換

フレーム	機能
CERTIFICATE	証明書と所有証明
CERTIFICATE_REQUEST	CertificateRequestメッセージ
CERTIFICATE_NEEDED	「証明書、どれ?」
USE_CERTIFICATE	「証明書、コレ!」

Certificate Compression

背景

- QUIC:
  - 接続確立とTLSハンドシェイクが並行動作
  - アドレス検証済でないクライアントにサーバ証明書を送りたい
  - リフレクション攻撃に使えない大きさに圧縮したい

Certificate Compression

- gzipまたはbrotliで証明書チェインを圧縮
- brotliの場合:
  - 中央値: 30%
  - 95パーセンタイル: 48%
- パケットに入る確率
  - 2パケット: 2% -> 54%
  - 3パケット: 55% -> 97%
- 「3倍」は許容可能な増幅率か?な?

https://datatracker.ietf.org/meeting/101/materials/slides-101-tls-sessa-certiicate-compression-00

SNI Encryption

プライバシー保護の進捗状況

- DNS暗号化:
  - DNS over TLS (2016)
  - DNS over HTTPS (WG Last Call)
    - 関連プロトコル: TLS 1.3 0-RTT, HTTP/2, QUIC
- SNI暗号化: ???
- 証明書暗号化: TLS 1.3
- ユーザトラッキングの抑止
  - セッションチケットのワンオフ化: TLS 1.3
  - IPアドレス、ポート番号の変更: QUIC

検討されている解決策

- SNIを暗号化
  - 暗号化するための鍵の配布方法が問題
- TLS over TLS
  - ２重暗号化のオーバーヘッド
- altsvc + Secondary Certificates for HTTP/2
  - HTTP/2専用

所感

所感

- TLSのメジャーバージョンアップは完了
  - メジャーな実装も対応済
  - 今後は周辺の整備が焦点に?
- プライバシー保護とOssification対策は重要なテーマ