QUIC の仕組み

QUIC: A UDP-Based Multiplexed and Secure Transport (RFC9000)

- 1つの Connection 内に複数の Stream (Multiplexed) → HoL を排除
- 特殊な場合を除き、原則暗号化される (Secure)
  TCP と TLS で別々に行われていたハンドシェイク等をまとめて行う
- UDP 上で QUIC 独自の輻輳制御や再送を行い信頼性を確保 (Transport)

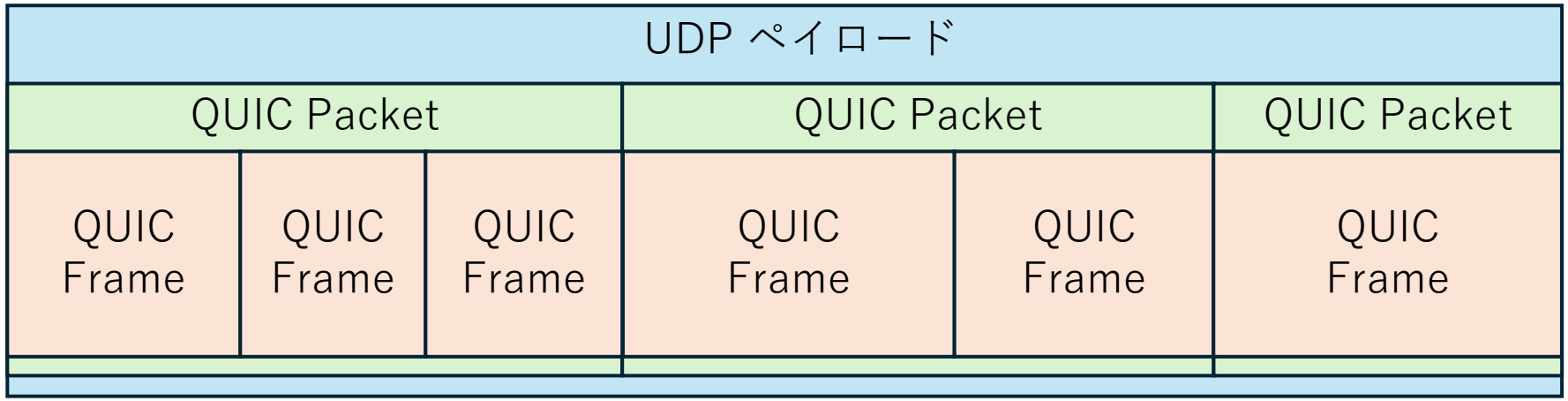
その他の特徴

- IP アドレスに依拠しない Connection の管理
  → IP アドレスが変わっても接続が切れない
- ユーザーライブラリとして実装される
  → 各自が自由に拡張・改良可能
  ※ 相互接続性の保証が大変な気がする

QUIC Packet, Frame

- QUIC は1つのUDPパケットに複数の “QUIC Packet”を持つ
  - Initial Packet, Handshake Packet, 0-RTT Packet 等
  - 各 Packet は Packet Number (PKN) を持つ
- 各 QUIC Packet 内は複数の “QUIC Frame” を持つ
  - ACK Frame, PADDING Frame, CRYPTO Frame, STREAM Frame 等

→ **無駄な UDP パケットや Round Trip の削減**



QUIC のハンドシェイク

QUIC におけるハンドシェイクは2段階

Connection のハンドシェイク

- 暗号(TLS ベース) やQUICに関するパラメータの交換
- Connection ID の割り当て
- 暗号化に利用するシークレットの導出

Streamのハンドシェイク

- Connection 内でそれぞれ独立したデータチャネル(Stream) を確立
- Bi-directional, Uni-directional など種類がある

QUIC のハンドシェイク(簡易版)

RFC 9000 Section 7.1. Example Handshake Flows より抜粋

```
Client                                                  Server

Initial[0]: CRYPTO[CH] ->

                                     Initial[0]: CRYPTO[SH] ACK[0]
                           Handshake[0]: CRYPTO[EE, CERT, CV, FIN]
                                     <- 1-RTT[0]: STREAM[1, "..."]

Initial[1]: ACK[0]
Handshake[0]: CRYPTO[FIN], ACK[0]
1-RTT[0]: STREAM[0, "..."], ACK[0] ->

                                              Handshake[1]: ACK[0]
                  <- 1-RTT[1]: HANDSHAKE_DONE, STREAM[3, "..."], ACK[0]
```

Figure 5: Example 1-RTT Handshake

QUIC ハンドシェイクの特徴

Packet の種類ごとに独立した PKN の空間を持つ

- それぞれのハンドシェイクが独立している
- それぞれ対応する Packet ごとに ACK を返す

1-RTT で Stream を確立しデータを流せる

- 暗号のハンドシェイクと Stream の送信を同じ UDP パケットで行う
- TCP + TLS (3-way + 1-RTT) より圧倒的に RTT の回数が少ない
- Session Resumption の場合は 0-RTT で送ることも可能

QUIC の暗号

QUIC では**全ての Packet のペイロード**が暗号化される

- ヘッダ部は暗号化されないが、PKN は “Header Protection” で保護される
- ペイロードの暗号化では、**ヘッダの認証も同時に行われる**

```
Handshake Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 2,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

部分的に**保護**

暗号化

Figure 17: Handshake Protected Packet

RFC9000 Section 17.2.4. Handshake Packet より

```
1-RTT Packet {
  Header Form (1) = 0,
  Fixed Bit (1) = 1,
  Spin Bit (1),
  Reserved Bits (2),
  Key Phase (1),
  Packet Number Length (2),
  Destination Connection ID (0..160),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

部分的に**保護**

暗号化

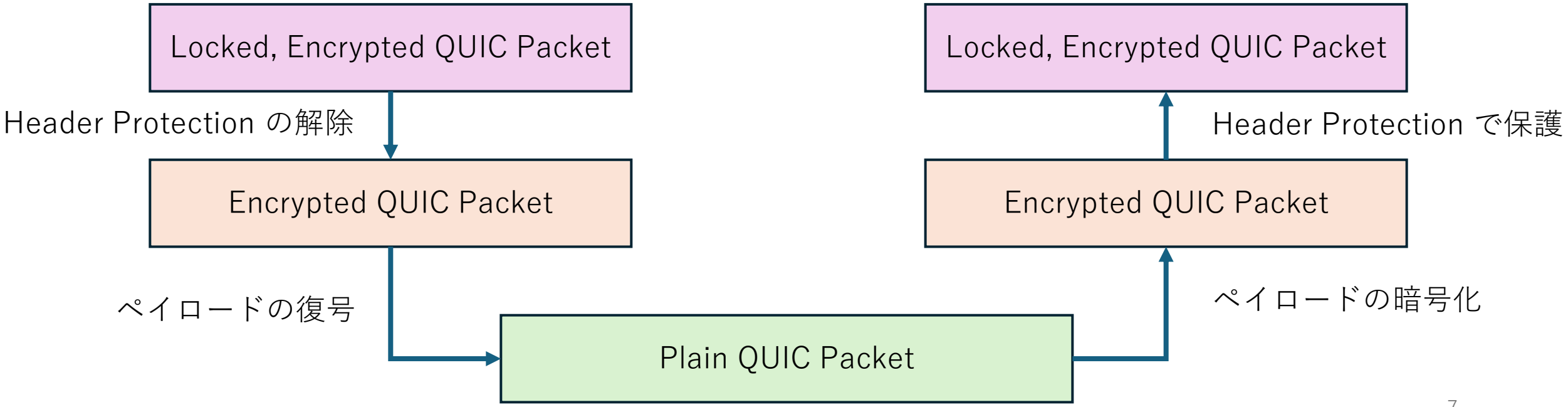
Figure 19: 1-RTT Packet

RFC9000 Section 17.3.1. 1-RTT Packet より

QUIC Packet の暗号化・復号

QUIC Packet の暗号化・保護は2段階

- QUIC Header Protection による保護 → ヘッダの保護
- AEAD(認証付き暗号) による暗号化 → ペイロードの保護



QUIC Header Protection

PKN やヘッダのフラグを保護する

- 暗号化された Payload を利用した**マスク**と XOR する

→ **Middle Box 等による介入を抑止する**

```
Initial Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 0,
  Reserved Bits (2),         # Protected
  Packet Number Length (2),  # Protected
  Version (32),
  DCID Len (8),
  Destination Connection ID (0..160),
  SCID Len (8),
  Source Connection ID (0..160),
  Token Length (i),
  Token (..),
  Length (i),
  Packet Number (8..32),     # Protected
  Protected Payload (0..24), # Skipped Part
  Protected Payload (128),   # Sampled Part
  Protected Payload (..)     # Remainder
}
```

```
1-RTT Packet {
  Header Form (1) = 0,
  Fixed Bit (1) = 1,
  Spin Bit (1),
  Reserved Bits (2),         # Protected
  Key Phase (1),             # Protected
  Packet Number Length (2),  # Protected
  Destination Connection ID (0..160),
  Packet Number (8..32),     # Protected
  Protected Payload (0..24), # Skipped Part
  Protected Payload (128),   # Sampled Part
  Protected Payload (..),    # Remainder
}
```

Figure 7: Header Protection and Ciphertext Sample

RFC9001 Section 5.4.1. Header Protection Application より

QUIC Header Protection

マスクはシークレットから導出される鍵(HP key) で
Payload を暗号化して導出する

- マスクは AES-ECB を用いる場合と ChaCha20 を用いる場合がある
  使い分けはハンドシェイクで選択される AEAD アルゴリズムに依る

```
mask = header_protection(hp_key, sample)

pn_length = (packet[0] & 0x03) + 1
if (packet[0] & 0x80) == 0x80:
   # Long header: 4 bits masked
   packet[0] ^= mask[0] & 0x0f
else:
   # Short header: 5 bits masked
   packet[0] ^= mask[0] & 0x1f

# pn_offset is the start of the Packet Number field.
packet[pn_offset:pn_offset+pn_length] ^= mask[1:1+pn_length]
```

Figure 6: Header Protection Pseudocode

RFC9001 Section 5.4.1. Header Protection Application より

```
header_protection(hp_key, sample):
  mask = AES-ECB(hp_key, sample)
```

RFC9001 Section 5.4.3. AES-Based Header Protection より

```
header_protection(hp_key, sample):
  counter = sample[0..3]
  nonce = sample[4..15]
  mask = ChaCha20(hp_key, counter, nonce, {0,0,0,0,0})
```

RFC9001 Section 5.4.4. ChaCha20-Based Header Protection より

認証付き暗号(AEAD)

Authenticated Encryption with Associated Data(AEAD)

- **データの機密性と完全性(正しさ)を担保する**
- 復号時に認証を行い、失敗すると復号失敗とする

cipher, tag = AEAD_Enc(plain, ad, key, nonce)

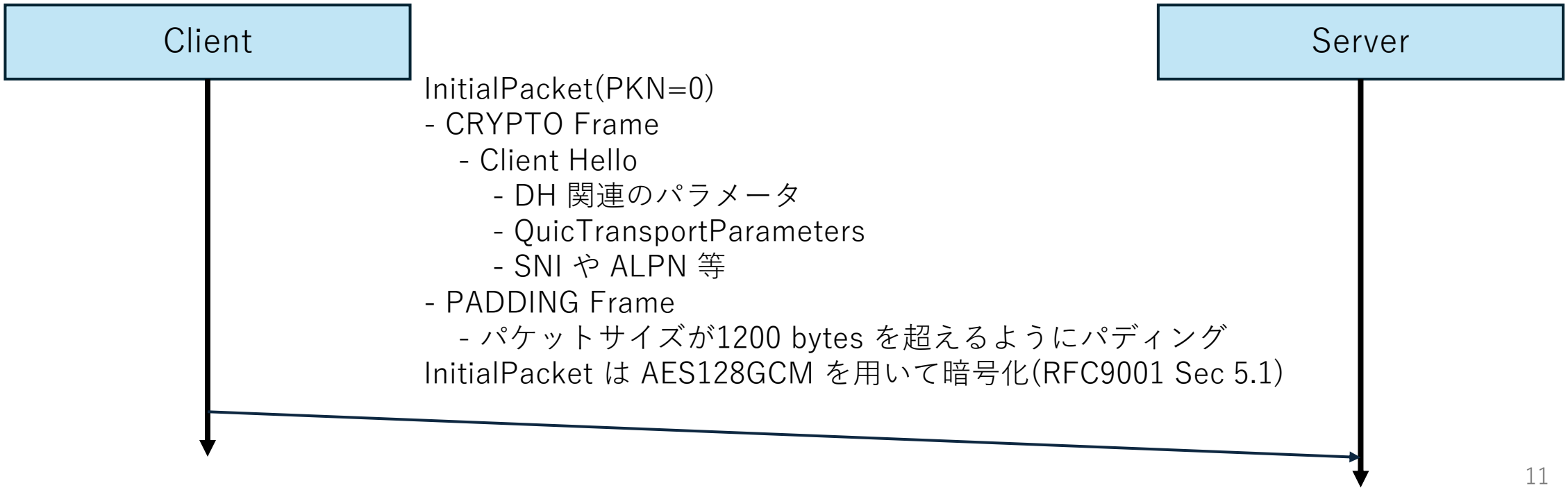
tag: 認証用のタグ, ad: 暗号化はされないが認証はされるデータ

plain = AEAD_Dec(cipher, tag, ad, key, nonce)

- QUIC では AES-GCM と Chacha20-Poly1305 が主に利用される

実際のハンドシェイク

QUIC では QUIC Packet を**最初から**暗号化する
暗号部のハンドシェイクは TLS 1.3 + QUIC 拡張(RFC9001)



Initial Secret

Initial Packet は “Initial Secret” から導出された鍵を用いて暗号化

- QUIC のバージョンごとに指定される salt と
  クライアントから見た宛先 Connection ID を基に生成
- 宛先 Connection ID はこの時点ではクライアントはランダムに生成
- 生成した secret から更に、HKDF(RFC5869) を用いてlabel を “quic key”, “quic iv”, “quic hp” として鍵、初期ベクトル、Header Protection 用鍵を生成する
  ※ TLS 1.3 の鍵導出でラベルが変わっただけ
- AEAD で利用する nonce は iv と PKN の XOR した結果を用いる(key はそのまま用いる)

```
initial_salt = 0x38762cf7f55934b34d179ae6a4c80cadccbb7f0a
initial_secret = HKDF-Extract(initial_salt,
                              client_dst_connection_id)

client_initial_secret = HKDF-Expand-Label(initial_secret,
                                          "client in", "",
                                          Hash.length)
server_initial_secret = HKDF-Expand-Label(initial_secret,
                                          "server in", "",
                                          Hash.length)
```

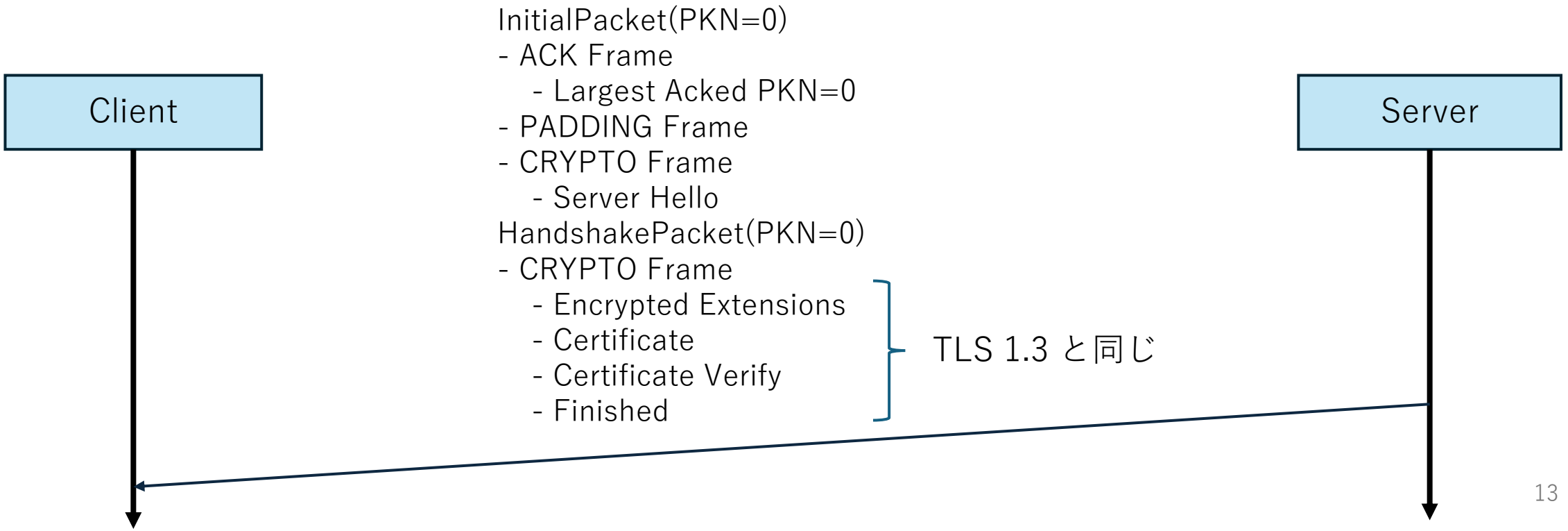
RFC 9001 Section 5.2. Initial Secrets

実際のハンドシェイク

サーバーからの Initial Packet を受け取る

同時に Handshake Packet も受け取る

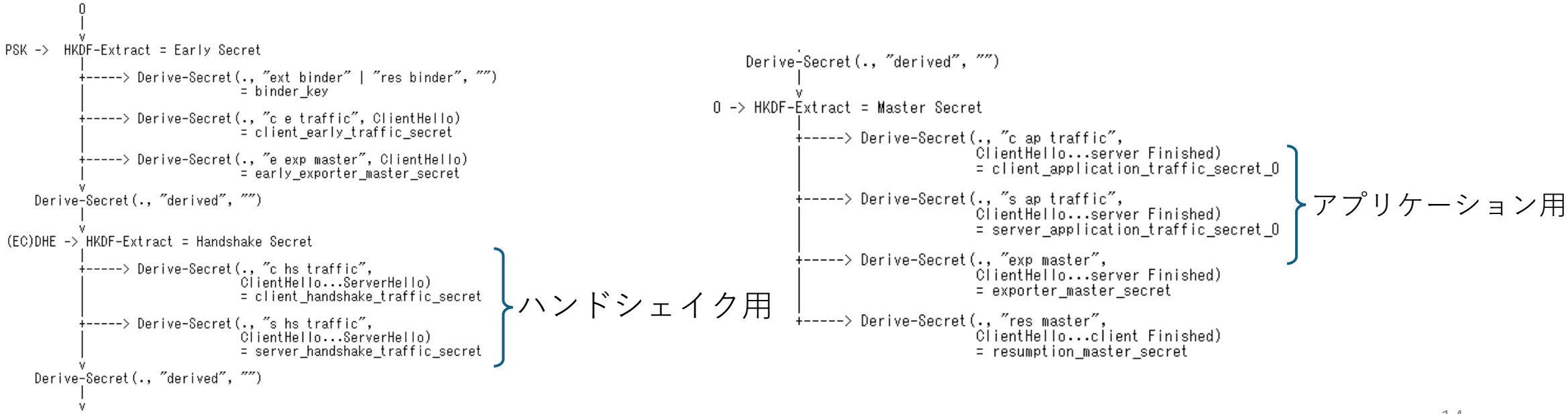
NEW_CONNECTION_ID Frame を含む 1-RTT Packet を受け取ることもある



Handshake Secret

Handshake Packet は Handshake Secret から導出される鍵を用いる

- Handshake Secret は TLS 1.3 における handshake secret と同じ
  ＝ ClientHello, ServerHello から導出する
- key, iv, hp 生成に利用するラベルは “quic key”, “quic iv”, “quic hp”



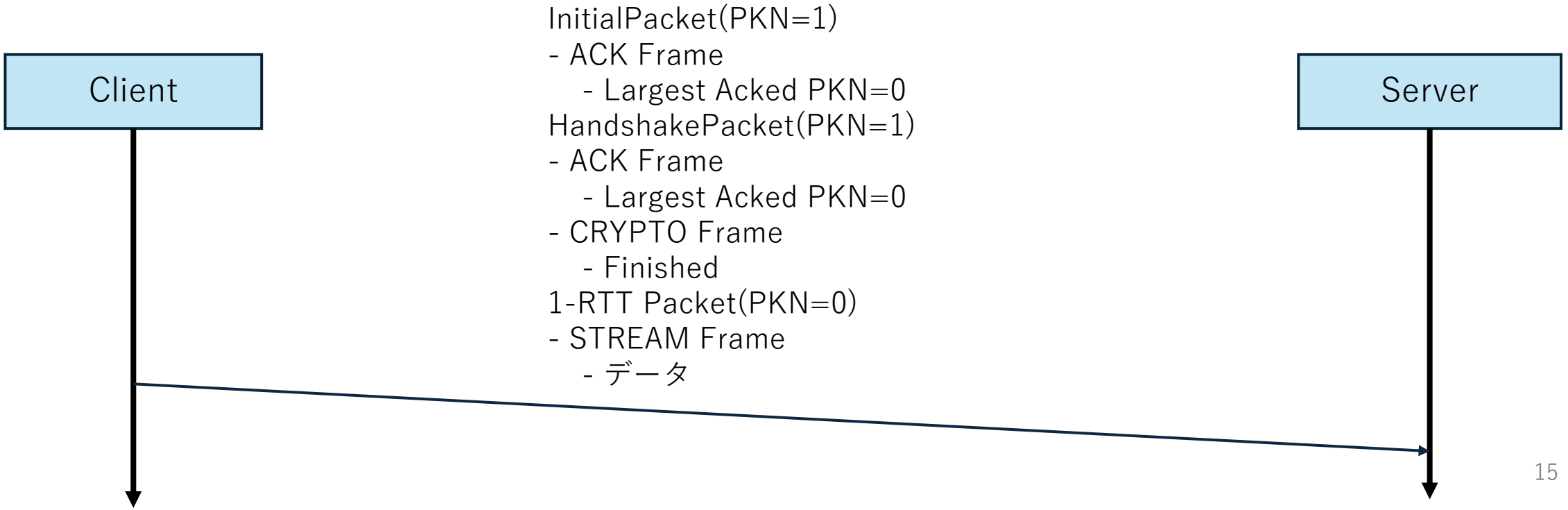
RFC8446, Section 7.1 Key Schedule より

実際のハンドシェイク

InitialPacket(PKN=0) に対する ACK

HandshakePacket(PKN=0) に対する ACK, Client Finished

必要に応じて STREAM Frame を 1-RTT Packet につめて送信



Application Secret

STREAM Frame を含む 1-RTT Packet で用いる Secret

- Application Secret は TLS 1.3 における application secret と同じ = CH, SH, EE, CT, CV, SV FIN から導出する
- key, iv, hp 生成に利用するラベルは “quic key”, “quic iv”, “quic hp”
- Key の Update が行われることがある
  - 新旧の鍵は Key Phase bit を用いて判断する
  - 双方ともに新しい鍵に移行したら、Update 完了
  - 詳細は RFC9001 Section 6. Key Update を参照のこと