Hybrid Post-Quantum Key Exchange for TLS 1.3

栗原 淳

2024-12-19

はじめに

この資料の位置付け

2024 年現在、ドラフト提案がなされている TLSv1.3 において耐量子公開鍵暗号による (共通) 鍵交換を実現する規格

- Hybrid Key Exchange in TLS 1.3 [1]
- Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3[2]

について、栗原自身の理解を深めるとともに、技術者にざっくり解説することを目的とする。

[1] https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/
[2] https://datatracker.ietf.org/doc/draft-kwiatkowski-tls-ecdhe-mlkem/

公開鍵暗号と量子コンピュータ

古典的な公開鍵暗号:

- TLS をはじめとするインターネット基盤の根幹に利用
- 素因数分解問題・離散対数問題の求解困難性に安全性が依存

量子コンピュータ

- 素因数分解・離散対数問題は，Shor のアルゴリズム [3] で求解可能
- N 量子ビットコンピュータは、1 回の演算で 2^N 回の演算を実行[4]
  → 1 量子ビットの増加で計算速度が 2 倍

古典コンピュータは引き続き発展していく、すなわち現在の公開鍵暗号の強化も進むだろうが、量子コンピュータの発展速度は、その安全性強化のペースを上回ると予測。[5]

[3] P.W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” in Proc. FOCS, 1994, pp. 124–134.

[4] 古典コンピュータは 2^N 回の演算は 2^N 回の逐次実行が必要

[5] NIST は、2048 ビット RSA 暗号は 2030 年頃に危殆化と予測。現在は 4096 ビットの利用が主だが、2030 年には一体何ビットを使えばいいのか？ 現実的に超長ビットの RSA なんて動くのか？

では、量子コンピュータが現実的になったら対応すればいいのか？

Harvest now, decrypt later (HNDL) 攻撃

「暗号化データを収集・保存しておき、あとで復号」という、リアルタイム性はないものの、(量子) 計算機の発展を待った有効な攻撃の思想。[6]

⇓

今現在、ネットワーク上を流れるデータを収集・保存している攻撃者がいた場合[7]、それはいずれ解読される可能性が高い

近い将来に訪れる量子コンピュータの発展を待たずして、対策が必須
⇒ 量子アルゴリズムでも効率的に解けない 耐量子公開鍵暗号 の開発・標準化が一気に進むことに

というわけで、待ったなし。

[6] 「たとえ一部が解読されても、他の暗号化データには影響させない」という Perfect Forward Secrecy とも関連する。

[7] 国家規模では存在。米国政府機関は実際に SSL/TLS の暗号化データを保存していたことが、2013 年のスノーデン事件で暴露された。

ML-KEM

耐量子公開鍵暗号の標準化

米国 NIST が、耐量子公開鍵暗号 (暗号化/鍵交換と、署名アルゴリズム) の募集・標準化検討を 2016 年より開始、2018 年 1 月～Round 1 選考、2019 年 1 月～Round 2 選考、2020 年 7 月～Round 3 選考を実施。

2022 年 7 月に Round 3 の Finalist を決定、それらをベースに 2024 年 8 月に以下の 3 つ [8] を NIST 標準として発表。

- **Module-Lattice-Based Key-Encapsulation Mechanism Standard (FIPS 203: ML-KEM)**[9] → **鍵カプセル化アルゴリズム**
- **Module-Lattice-Based Digital Signature Standard (FIPS 204: ML-DSA)**[10] → 電子署名アルゴリズム
- **Stateless Hash-Based Digital Signature Standard (FIPS 205: SLH-DSA)**[11] → 電子署名アルゴリズム

[8] Round 3 の Finalist は 4 つだが、4 つ目 (FALCON) は現在文書策定中 (FIPS 206: FN-DSA)

[9] https://csrc.nist.gov/pubs/fips/203/final

[10] https://csrc.nist.gov/pubs/fips/204/final

[11] https://csrc.nist.gov/pubs/fips/205/final

NIST FIPS の元となった Round 3 Finalists

- SLH-DSA: SPHINCS$^+$ (ハッシュ関数ベース)
- ML-DSA: CRYSTALS-DILITHIUM (格子問題ベース)
- ML-KEM: CRYSTALS-KYBER (格子問題ベース)
- FN-DSA: FALCON (高速フーリエ変換ベース、FIPS 206 予定)

NIST PQC 標準化 Round 4

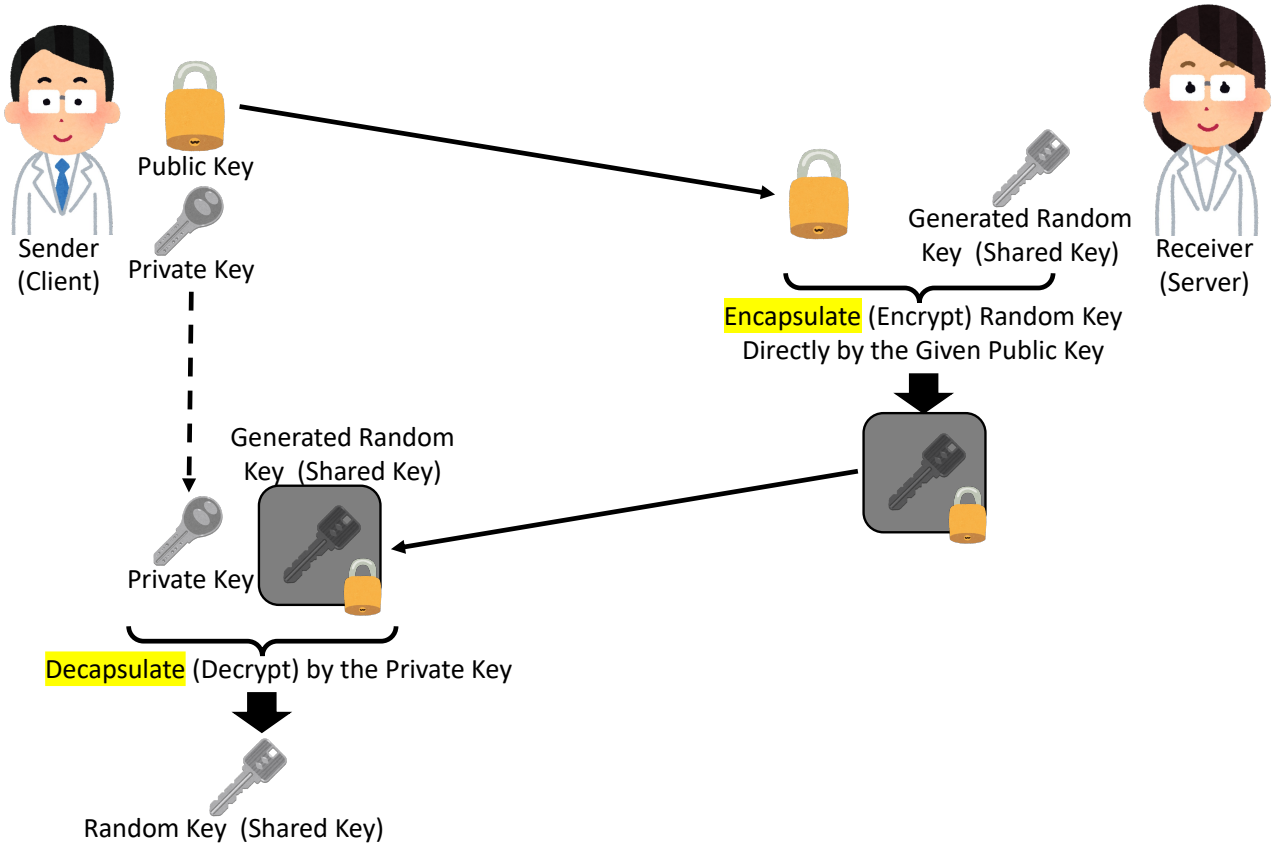
Round 2 の Finalist は、特に有望で先に標準化すべきと判断されたアルゴリズムと、それ以外の 2 種類に分類された。このため、Round 3 では前者を中心に評価が行われたようである。
Round 3 終了後、2022 年 7 月より Round 4 が開始され、残った Round 2 Finalist のうち 4 つ [12] を再度選定し、評価を開始している。

[12] 開始までに破られなかったもの

というわけで、ML-KEM に焦点を当てる

鍵カプセル化 (Key Encapsulation Mechanism) とは？

ナイーブな RSA 暗号化のように、**共通鍵を暗号化するための、公開鍵暗号方式のこと。**[13]

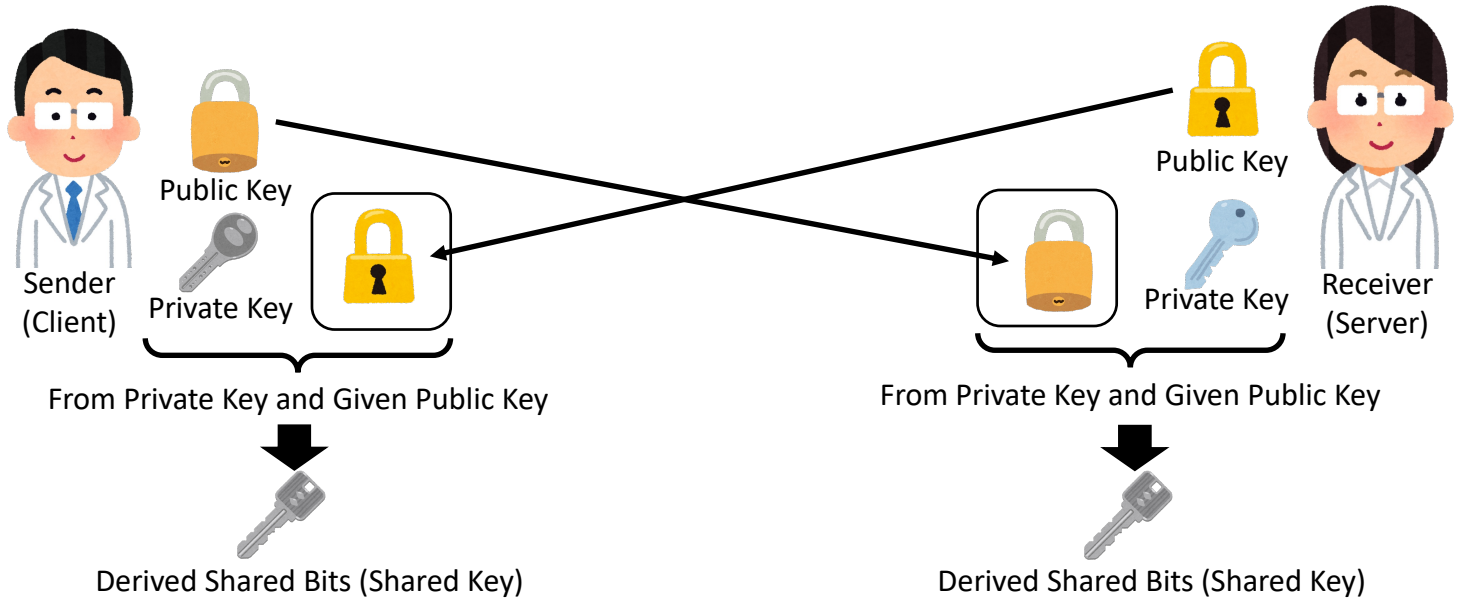


Key Encapsulation Mechanism (KEM) の手順

受信者が生成したランダム共有鍵を、送信者の公開鍵で暗号化して送付

[13] Diffie-Hellman 鍵交換は、送受信者間で「共通鍵を暗号化して送って共有」するのではなく、「同じ共通鍵を独立に作る (鍵交換)」手法。一方、RSA では直接暗号化できる。

現行の TLS 1.3 の規格[14] では、公開鍵暗号方式による鍵交換として、Forward Secrecy を有する (EC)DHE[15] 方式のみが規定されている。[16]



TLS で従来用いられる (EC) Diffie-Hellman 鍵交換 ((EC)DH KX) の手順
自身の秘密鍵と相手の公開鍵から Shared Bits を導出

---

[14] RFC8446 https://www.rfc-editor.org/rfc/rfc8446

[15] (Elliptic Curve) Diffie-Hellman Ephemeral: 毎回鍵ペアを生成・使い捨てる Ephemeral 方式

[16] 固定のサーバ RSA 鍵による鍵の共有は、「Bleinchenbacher 攻撃に脆弱」という理由から、TLS 1.3 では削除された。おそらく、「RSA 鍵の毎回の生成は低速」もある。RSA 署名のついた証明書は利用できることに注意。

紛らわしいので、以降の言葉の定義[17]

- **鍵交換 (Key Exchange; KX)**: (EC)DH 鍵交換方式
  - **公開鍵 Public Key**
  - **秘密鍵 Private Key**
- **鍵カプセル化 (Key Encapsulation Mechanism; KEM)**:
  ランダム共通鍵を暗号化して送付する方式
  - **カプセル化鍵 Encapsulation Key**: 暗号化に用いる公開鍵
  - **カプセル化解除鍵 Decapsulation Key**: 復号に用いる秘密鍵

[17] FIPS 203 ML-KEM および
`https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/` に準拠

ML-KEM の種類・鍵長・パフォーマンス

$k \in \{2, 3, 4\}$ に対し、ML-KEM-$256k$ を定義。鍵長は全てバイト単位。
ML-KEM-$256k$ は AES-$64k$ と同等のセキュリティ強度と見做される。(e.g., AES-192 = ML-KEM-768)

Table: ML-KEM における各データサイズ

	カプセル化鍵	カプセル化解除鍵	暗号文	共有鍵
ML-KEM-1024	1,568	3,168	1,568	32
ML-KEM-768	1,184	2,400	1,088	32
ML-KEM-512	800	1,632	768	32

データサイズは非常に大きいが X25519 (ECDH の一種) よりも高速な処理を実現

Table: Cloudflare による X25519 と ML-KEM の比較 [18]

	データサイズ (Bytes)		処理回数/秒	
	Client→Sever	Server → Client	Client	Server
ML-KEM-768	1,184 (カプセル化鍵)	1,088 (暗号文)	31,000 (鍵生成・復号)	70,000 (暗号化)
ML-KEM-512	800 (カプセル化鍵)	768 (暗号文)	50,000 (鍵生成・復号)	100,000 (暗号化)
X25519	32 (公開鍵)	32 (公開鍵)	17,000 (鍵生成・共有)	17,000 (鍵生成・共有)

[18] `https://blog.cloudflare.com/post-quantum-for-all/`, 計測環境は不明

ML-KEM の推奨鍵長や処理速度

既存の公開鍵暗号同様に、鍵長が長くなれば処理速度が低下する。
NIST は ML-KEM-768 (カプセル化鍵長 1184 bytes) を、セキュリティ・パフォーマンスの観点から推奨している

> *(Section 8, FIPS 203)*
> *NIST recommends using ML-KEM-768 as the default parameter set, as it provides a large security margin at a reasonable performance cost.*

Hybrid Post-Quantum Key Exchange/Agreement

利用・標準化が進む Hybrid Post-Quantum KX

ざっくり言うと：Hybrid Post-Quantum (PQ) Key Exchange/Agreement

- **(EC)DH KX** (e.g., X25519, secp256r1)
- **PQ KEM** (i.e., ML-KEM)

を両方・独立に実行し、2 つの共通鍵を共有。2 つの共通鍵は結合・攪拌されて 1 つの共通鍵として利用。

なぜ 2 つを同時に使うのか？

	Pros	Cons
(EC)DH KX	長い安全性検証の歴史	量子コンピュータに対して危殆化が懸念
PQ KEM	量子コンピュータで効率的な解法がない	浅い安全性検証の歴史

⇒ Hybrid = (EC)DH KX と PQ KEM が相互に安全性を補い合う

- PQ KEM (i.e., ML-KEM) を効率的に破る手法が発見されても、(EC)DH KX で安全性を担保。
- 量子コンピュータが急速に発展して (EC)DH KX が危殆化しても、PQ KEM で安全性を担保。

例えば以下の Hybrid PQ KX が IETF ドラフト提案されている [19]

- Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3[20]
- X-Wing: general-purpose hybrid post-quantum KEM[21]
- X25519Kyber768Draft00 hybrid post-quantum KEM for HPKE[22]

特に、**TLS 1.3 の ECDHE-MLKEM は、その前バージョンを含めてインターネット上での実装・利用が進んでいる。**[23] 例えば、

- ブラウザ: Firefox 123 以上、Chrome 124 以上
- CDN: Cloudflare・Google Cloud 等 [24]
- TLS ライブラリ: BoringSSL, Rustls

---

19 https://wiki.ietf.org/group/sec/PQCAgility

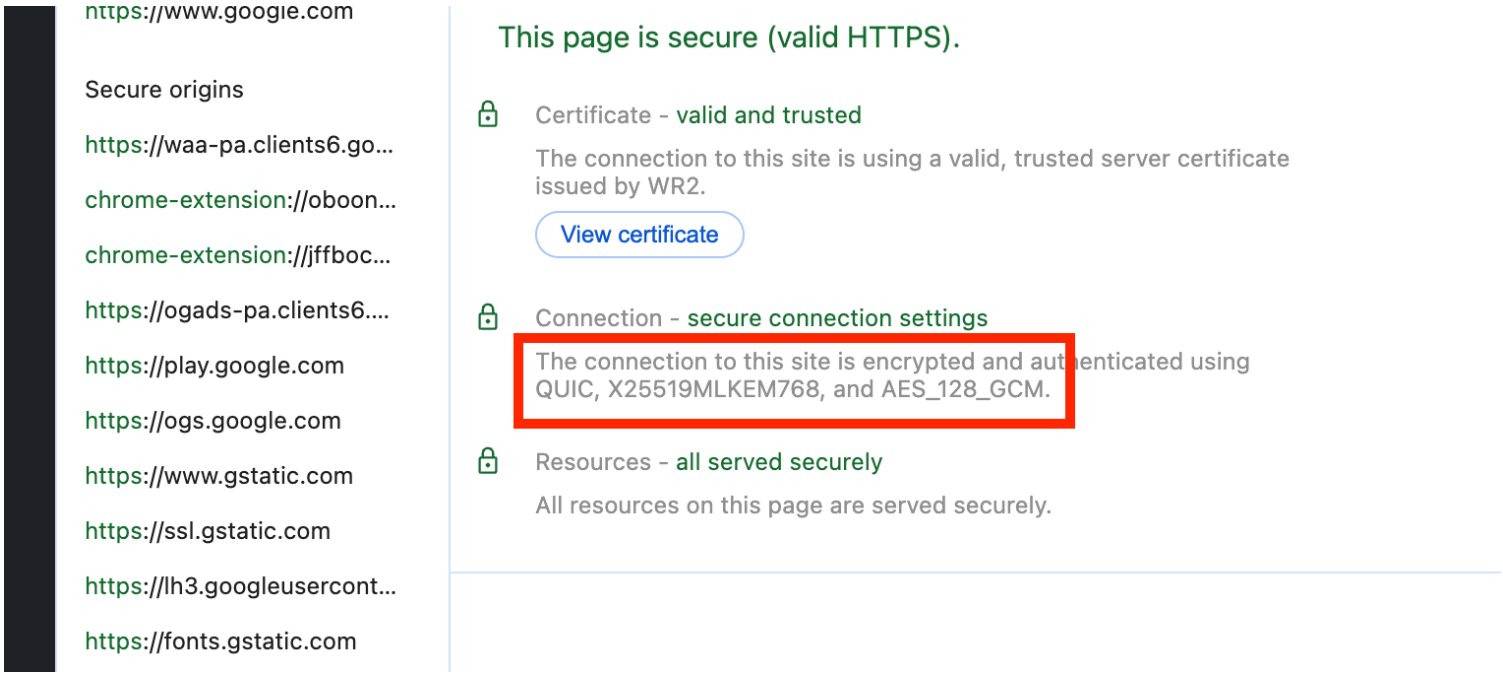
20 https://datatracker.ietf.org/doc/draft-kwiatkowski-tls-ecdhe-mlkem/

21 https://datatracker.ietf.org/doc/draft-connolly-cfrg-xwing-kem/

22 (Expired) https://datatracker.ietf.org/doc/draft-westerbaan-cfrg-hpke-xyber768d00/

23 前バージョン: X25519Kyber768Draft00。https://pq.cloudflareresearch.com を参照。

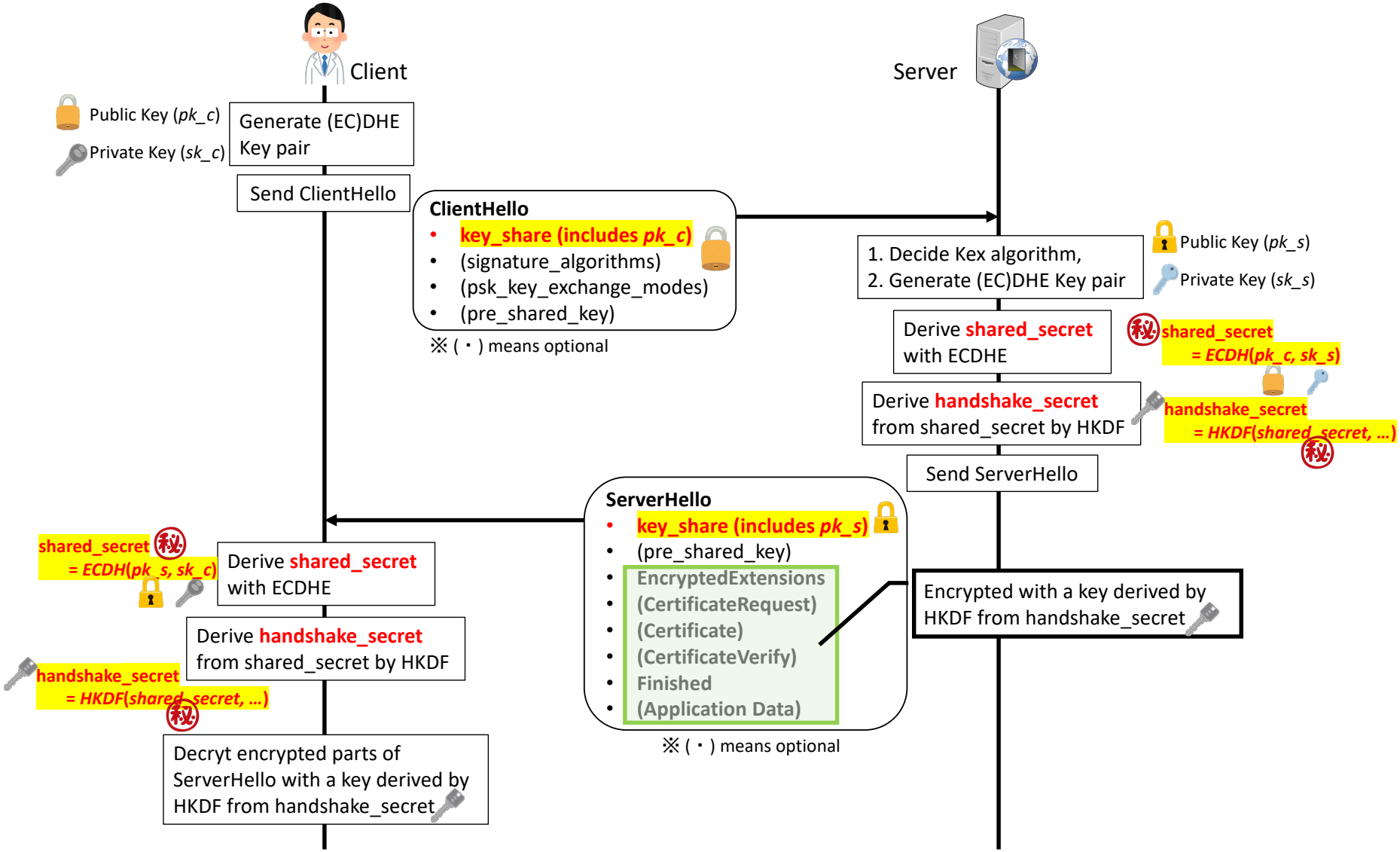
24 対応ブラウザであれば https://www.google.com へ Hybrid PQ KX で接続可能



Chrome の flags において”Use ML-KEM in TLS 1.3” を Enabled にした状態で、google.com へアクセス

以降、この「TLS 1.3 における Hybrid PQ KX」に焦点を当てる

TLS 1.3 の従来の鍵交換フロー



現状の TLS 1.3 では、Client/ServerHello の key_share 拡張で公開鍵を互いに送付。ECDH と HKDF による handshake_secret 鍵を導出。

Hybrid (PQ) KX を用いた TLS 1.3 での鍵交換

Hybrid Key Exchange in TLS 1.3

IETF RFC Draft[25] で策定中のフレームワーク。

- (EC)DHE など「Traditional」な鍵交換
- PQ KEM を含む新しい「Next-generation」な手法

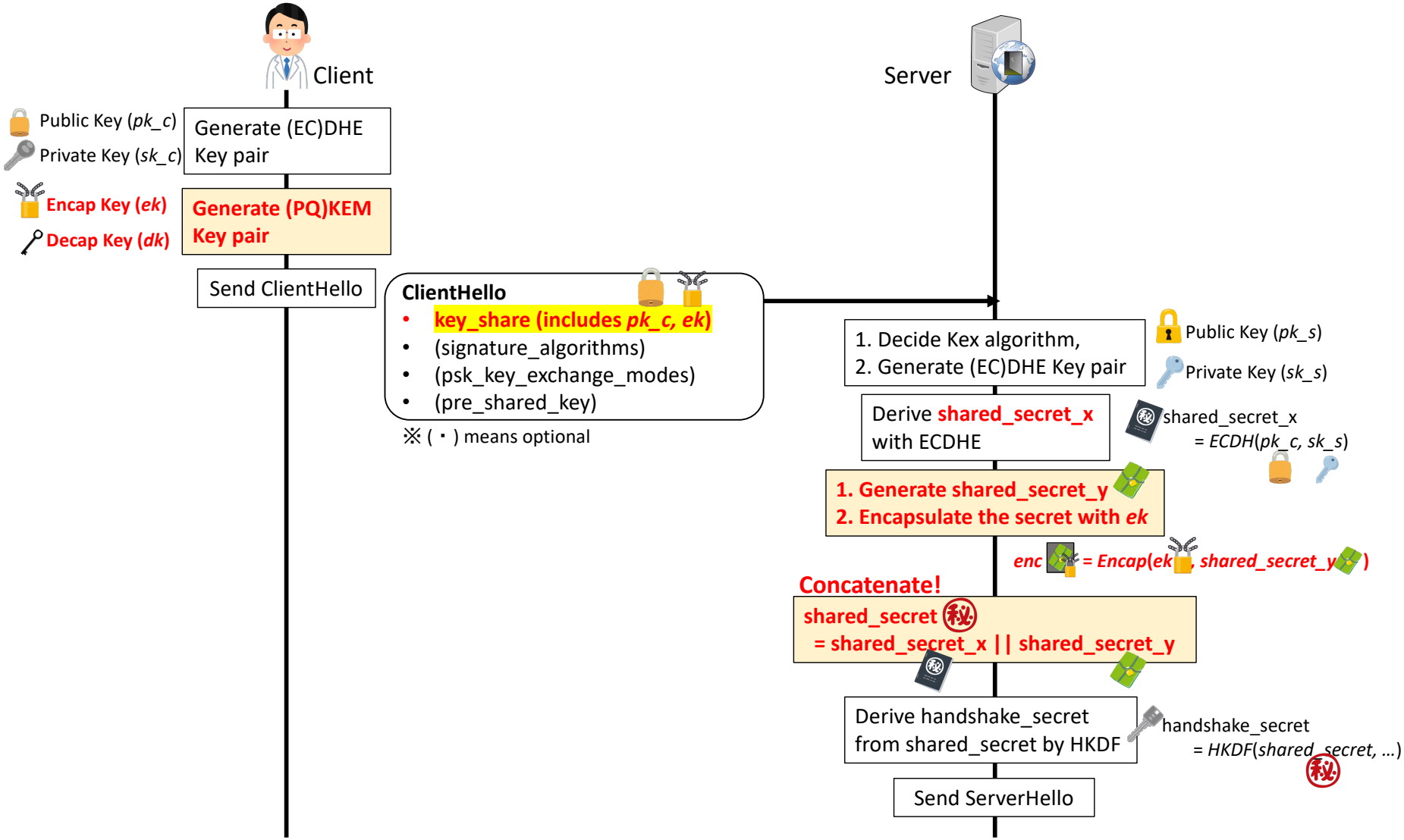
の両方を同時に実行することで、片方が破られてももう片方で TLS 接続の秘匿性を担保し続けることが目的。このフレームワークに ECDHE と ML-KEM を適用した手法 (後述) が、展開されつつある。

構造自体は全く単純で、2 つ (以上) の KX・KEM を同時に実行するだけ。KX・KEM で共有した鍵を全て結合して、従来通りに TLS 1.3 の接続確立の中で利用する。

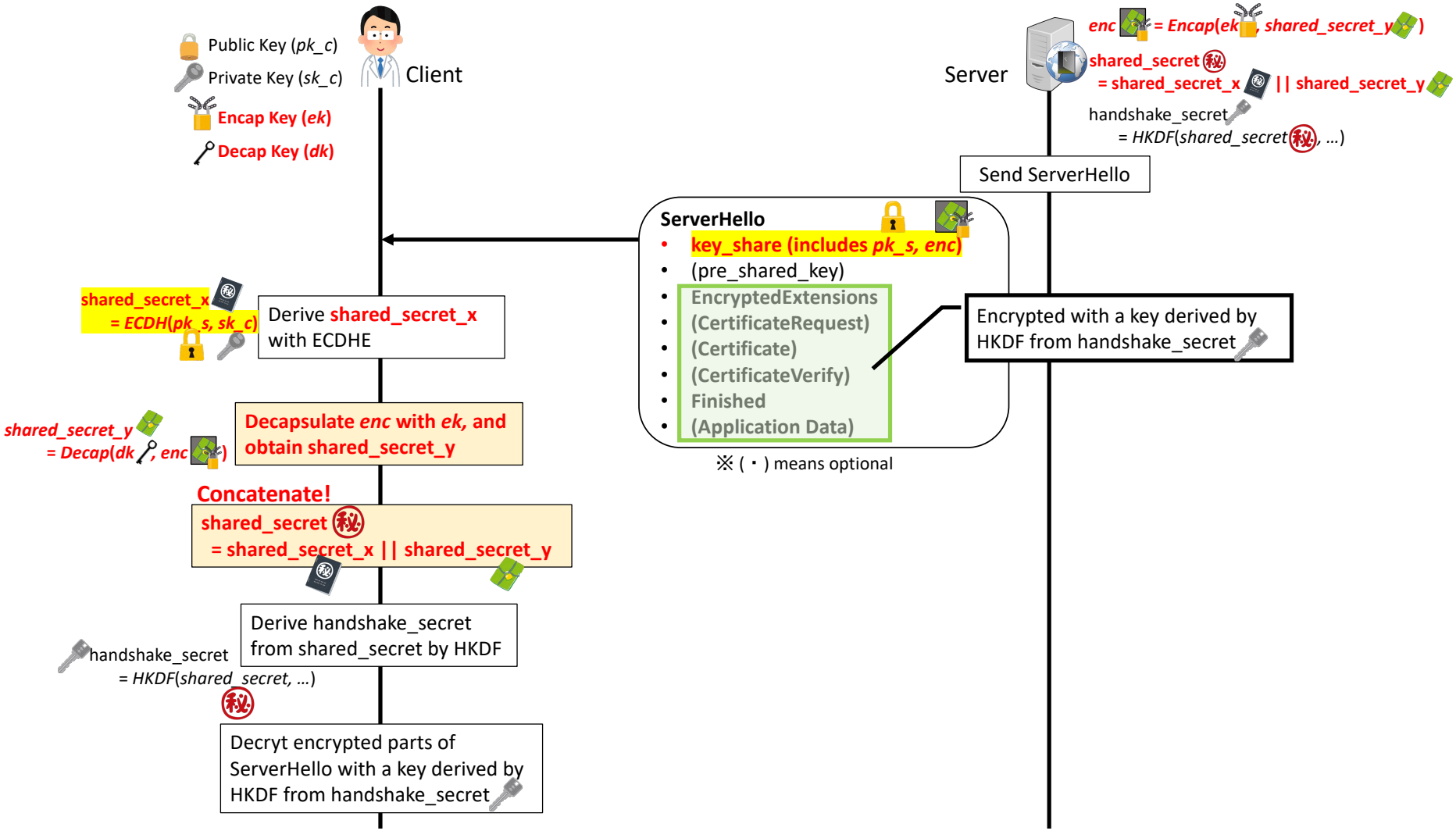
[25] https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/

TLS 1.3 Hybrid Key Exchange フロー

(EC)DHE と PQ KEM を仮定して、フローを紹介



1. ClientHello の key_share 拡張で新規の公開鍵・カプセル化鍵を同時送信
2. サーバは ECDH 鍵交換 (shared_secret_x 導出) と、ランダム生成鍵...y のカプセル化を実施、shared_secret_x と y を結合し shared_secret 導出、HKDF で handshake_secret 生成



3. ServerHello の key_share 拡張で新規の公開鍵・カプセル化した shared_secret_y を同時送信
4. クライアントは ECDH 鍵交換 (shared_secret_x 導出) と、カプセル化解除を実施 (..._y 導出)、shared_secret_x と y を結合し shared_secret 導出、HKDF で handshake_secret 生成

key_share が大きく更新された要素なので、それを見ていこう

Client/ServerHello の key_share 拡張詳細 (現行 TLS 1.3)

まずは TLS 1.3 そのものを復習する

key_share 拡張 [26]

エンドポイントの暗号パラメタを格納・送付するための
Client/ServerHello メッセージの拡張フィールド。以下を格納。

- **ClientHello: client_share (KeyShareClientHello 構造体)**

```
struct KeyShareClientHello {
  client_shares [KeyShareEntry] (パラメタの配列)
}
```

⇒ 前から順に優先度の高い鍵交換方式のパラメタを列挙

- **ServerHello: server_share (KeyShareServerHello 構造体)**

```
struct KeyShareServerHello {
  server_share KeyShareEntry (単一パラメタ)
}
```

⇒ client_share から 1 つパラメタを選択して、その返答のパラメタ

[26] https://datatracker.ietf.org/doc/html/rfc8446#section-4.2.8

KeyShareEntry

鍵交換方式の暗号パラメタを格納する構造体:

```
struct KeyShareEntry {
  group NamedGroup (暗号アルゴリズムの ID)
  key_exchange [u8] (シリアライズされたパラメタ)
}
```

⇒ アルゴリズム = group と、それに対するパラメタ = key_exchange (group 毎に構造が異なる) を指定

NamedGroup[27]

IANA[28] で合意・決定され、アルゴリズムを指定する 2 バイトの値。
現在の TLS 1.3 では以下が利用可能。

- ECDHE: secp256r1(0x0017), secp384r1(0x0018), secp521r1(0x0019), x25519(0x001D), x448(0x001E)
- DHE: ffdhe2048(0x0100), ffdhe3072(0x0101), ffdhe4096(0x0102), ffdhe6144(0x0103), ffdhe8192(0x0104)

[27] https://datatracker.ietf.org/doc/html/rfc8446#section-4.2.7

[28] Internet Assigned Numbers Authority, インターネットに関連する識別子を管理する機関

KeyShareEntry の key_exchange フィールドには何が入る？

group で決定される鍵交換方式の「パラメタ」:

- client_share (ClientHello) の場合: (EC)DHE 公開鍵のシリアライズされたバイナリ [29]
- server_share (ServerHello) の場合: (EC)DHE 公開鍵のシリアライズされたバイナリ [30]

現行の TLS1.3 では、KeyShareEntry は (EC)DHE 鍵交換のみ対応するため、いずれも公開鍵を送付する。DHE の場合は $y = g^x \pmod{p}$、[31]ECDHE の場合は、楕円曲線上のパラメタ。[32]

と、ここまでは現行の TLS 1.3 の復習。

では、**Hybird PQ KX の場合はどうなる？**

---

[29] https://datatracker.ietf.org/doc/html/rfc8446#section-4.2.8.1

[30] https://datatracker.ietf.org/doc/html/rfc8446#section-4.2.8.2

[31] x が秘密鍵。p のサイズまでパディングしてネットワークバイトオーダーで表す。

[32] secp***r1 の場合は点の座標 (x, y)、x***の場合は RFC7748 で定められる 32 バイト (x25519) もしくは 56 バイト (x448)

key_share に対応する方式のパラメタがないとき: HelloRetryRequest

TLS 1.3 では、クライアントが対応する NamedGroup 一覧を、ClientHello の **supported_group 拡張**で送付する。ただし、**client_share には、それら全ての公開鍵等のパラメタが送付されるとは限らない。**

⇓

もし対応する NamedGroup のパラメタが client_share に含まれない場合:

- サーバは ServerHello ではなく、**HelloRetryRequest** を返送。HelloRetryRequest は、ServerHello 同様に key_share 拡張を有する。
- ここに、ClientHello で通知された supported_group 拡張のうち、サーバが対応する方式名 (selected_group) を選択・格納することで、当該の方式のパラメタを含む client_share を持つ ClientHello の送信を促す。

Hybrid PQ KX での key_share 拡張への変更

以下の 2 点が追加変更されている。

1. **NamedGroup の新たな定義**
   ⇒ IANA 2 バイト識別子を新たに付与、識別子の名前は単純にアルゴリズム名連結
2. **key_exchange も 2 つのアルゴリズムのパラメタのバイナリ連結**

NameGroup の新しい定義

[新アルゴリズム名] = [アルゴリズム名 1][アルゴリズム名 2] と定義。
例えば **secp256r1mlkem768 = secp256r1 + mlkem768 (0x11EB)**。

key_exchange の変更点

NamedGroup のアルゴリズム名の連結順に暗号パラメタをバイナリ連結したものとして定義。ただし、KEM は KX とは異なるため、key_share 内の暗号パラメタは新たに定義：

- **ClientHello: KEM のカプセル化鍵を単純にシリアライズしたもの**
- **ServerHello: カプセル化したランダム共有鍵**。即ち、サーバはランダムに共有鍵を自身で生成，カプセル化 (暗号化) する。

つまり、例えば secp256r1mlkem768 の場合:

- client_share: [secp256r1 クライアント公開鍵]||[mlkem768 カプセル化鍵]
- server_share: [secp256r1 サーバ公開鍵]||[mlkem768 カプセル化共有鍵]

Security Consideration[35]

現行の TLS 1.3 の (EC)DHE では、**Ephemeral Key Pair の再利用**が行われることがあり得る。
これを考慮して、(EC)DHE と組み合わされて利用される KEM は、

- IND-CCA2[33] 安全を担保する,
- Fujisaki-Okamoto 変換 [34] などを行って CCA 安全を担保する

のいずれかにより、カプセル化鍵が再利用されても安全性が担保されることを必須としている。

---

[33] 適応的選択暗号文攻撃に対する識別不可能性

[34] E. Fujisaki and T. Okamoto, “Secure Integration of Asymmetric and Symmetric Encryption Schemes,” J. Cryptology, vol. 26, no. 1, pp. 80–101, Dec. 2011.

[35] `https://datatracker.ietf.org/doc/html/draft-ietf-tls-hybrid-design-10#name-security-considerations`

ECDHE-MLKEM

ようやく具体的なパラメタのお話。
“X25519Kyber768Draft00”[36] のアップデート版。Kyber との違いは Appendix を参照。

ECDHE-MLKEM[37]

前述の “Hybrid Key Exchange in TLS 1.3” のフレームワークで動作する，以下の 2 種類の Hybrid PQ KX を規定

- **X25519MLKEM768** (ECDH: X25519 + PQ-KEM: ML-KEM-768)
  → key_share:
  NamedGroup: `mlkem768x25519` (0x11EC)
  key_exchange: [ML-KEM768 の公開鍵 or カプセル化共有鍵]||[X25519 公開鍵]
- **SecP256r1MLKEM768** (ECDH: secp256r1 + PQ-KEM ML-KEM-768) → key_share:
  NamedGroup: `secp256r1mlkem768` (0x11EB)
  key_exchange: [SecP256r1 公開鍵]||[ML-KEM768 公開鍵 or カプセル化共有鍵]

[36] https://datatracker.ietf.org/doc/draft-tls-westerbaan-xyber768d00/
[37] https://datatracker.ietf.org/doc/draft-kwiatkowski-tls-ecdhe-mlkem/

NamedGroup mlkem768x25519、X25519MLKEM768 ではないの？

NIST SP800-56r2 では、「**複数のアルゴリズムを並べるときには、NIST FIPS で認められたものを先頭にする**」というルールが定められている。このため、NIST FIPS 203 の ML-KEM を先にしている。(X25519 は IETF RFC ではあるが、実は NIST 標準じゃない)

Hybrid じゃない、ML-KEM 単体での鍵交換もドラフト提案されている

`https://datatracker.ietf.org/doc/draft-connolly-tls-mlkem-key-agreement/` にてドラフト提案中。IANA NamedGroup 2 バイト識別子は以下の通り。

- mlkem512 0x200
- mlkem768 0x201
- mlkem1024 0x202

Hybrid 方式の ML-KEM 部分のみを動作させるだけ。

Hybrid PQ KX/ECDHE-MLKEM のデメリット

もちろん，これを実装するには現行のプロトコルからのアップデートが大変という問題は存在。しかし、加えて **ClientHello/ServerHello 一気に大きくなる**というデメリットが存在。[38]

⇓

ServerHello はまだしも [39]、ClientHello のパケットが下位レイヤでフラグメント化される可能性がある。

⇓

中間のスイッチでは予期しない動作を引き起こす可能性もあるし、分割・結合によるパフォーマンスロスも無視できないかもしれない (が、解はない)。

---

[38] ML-KEM-768 の公開鍵は 1184 バイト、カプセル化共有鍵は 1088 バイト。一方で、X25519 公開鍵は 32 バイト……実に 30 倍以上。

[39] 証明書を含む場合は元々デカい

まとめ

その他の PQC の IETF 標準化状況

- Post-Quantum Cryptography Recommendations for Internet Applications (IETF ドラフト)[40]: PQC を使うインターネット上のプロトコルのベストプラクティス
- Terminology for Post-Quantum Traditional Hybrid Schemes (IETF ドラフト)[41]: Traditional な手法と Post-Quantum な手法のハイブリッドプロトコル全般に対する言葉の定義
- PQ/T Hybrid Key Exchange in SSH (IETF ドラフト)[42]: SSH 向け、ML-KEM と ECDHE のハイブリッド鍵共有方法 (PQ/T: Post-Quantum/Traditional)
- ML-KEM Post-Quantum Key Agreement for TLS 1.3 (IETF ドラフト)[43]: 前述した通り。

[40] https://datatracker.ietf.org/doc/draft-reddy-uta-pqc-app/
[41] https://datatracker.ietf.org/doc/draft-driscoll-pqt-hybrid-terminology/
[42] https://datatracker.ietf.org/doc/draft-kampanakis-curdle-ssh-pq-ke/
[43] https://datatracker.ietf.org/doc/draft-connolly-tls-mlkem-key-agreement/

まとめ

- 「Hybrid」な Post-Quantum Key Exchange/Agreement について説明した。
- TLS 1.3 でのドラフト提案 (多分このまま標準化される) について、現行からの変更点について説明した。
- Hybrid な方法は「Transitional (過渡期)」な方法かもしれないが、向こう 20 年くらいは使われ続けそう。理解しておいて損はない。

最後に手前味噌な紹介

ECDHE-MLKEM for TLS 1.3/QUIC 対応の、Rust 製 HTTPS リバースプロキシをリリースした。かなり高速。

`https://github.com/junkurihara/rust-rpxy`

Appendix

Kyber から ML-KEM への変更点[44]

- KEM での共有鍵生成時の 32 バイトの入力シード m を，NIST 標準準拠の乱数の入力へ変更
- 共有される鍵を、SHAKE 関数を掛けない、固定長 32 バイトへ変更
- カプセル化・解除時の入力フォーマットチェックを必須化
- カプセル化鍵ペアの生成シードに、格子の次元を含めて、利用ミスを防ぐように変更

[44] Appendix C, FIPS 203．および参考: 【耐量子計算機暗号・PQC】ML-KEM（CRYSTALS-KYBER）標準について: https://zenn.dev/ankoma/articles/f165d06efb1468

入力シード: m，カプセル化鍵: ek，カプセル化関数: Enc
共通鍵 K は以下のように導出。(式中の r は 32 バイト固定)

- CRYSTALS-KYBER: SHAKE[45] による可変長出力、m のランダム性が低いことを考慮してハッシュ化

$$\begin{aligned}\bar{K}||r &= H_{\mathrm{SHA3-512}}(H_{\mathrm{SHA3-256}}(m)|H_{\mathrm{SHA3-256}}(ek)),\\ c &= Enc(ek, m, r),\\ K &= SHAKE(\bar{K}||H_{\mathrm{SHA3-256}}(c))\end{aligned}$$

- ML-KEM: K は 32 バイト固定、m は NIST 標準乱数

$$K||r = H_{\mathrm{SHA3-512}}(m|H_{\mathrm{SHA3-256}}(ek))$$

[45] SHAKE: SHA3 規格の一種で、可変長のハッシュ値を出力