大和証券グループで行った実証実験の補足資料

検証環境を使った基礎検証の補足

1. 検証環境

検証に使用した端末は Amazon EC2 のインスタンスであり、その詳細は以下の通り：

項目	内容
インスタンスタイプ	c7i.large
AMI	ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-20251022
OS	Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1018-aws x86_64)
vCPU	2
メモリ	4GiB

また、個別のアルゴリズムの測定については以下のライブラリを使用した：

ライブラリ名	概要
Cryptography (Python)	バージョンは 46.0.3。 RSA、X25519、P-256 の性能を測定するために使用した。
liboqs	バージョンは 0.14.0。 各耐量子計算機暗号アルゴリズムの性能を測定するために使用した。 また、ラッパーとして liboqs-python を使用した。

更に、具体的な通信においては以下のライブラリを使用した：

ライブラリ名	概要
OpenSSL	バージョンは 3.5.4。 TLS 通信のためのクライアントやサーバとして使用した。
OQS Provider	バージョンは 0.10.1-dev。 一部の耐量子計算機暗号を用いた TLS 通信のために使用した。

尚、クライアント側の実装のために pwntools や paramiko も使用しているが、ハンドシェイクとは無関係な通信にのみ用い、TLS 通信は OpenSSL を使用して行っている。

2. 留意事項

尚、各ライブラリのバージョンは実験開始当時（2025 年 11 月時点）の最新版である。

執筆時点に於いて、cryptography は 46.0.5、liboqs は 0.15.0、OpenSSL は 3.6.1/3.5.5、OQS Provider は 0.11.0 が最新のバージョンとなっている。

また、OpenSSL は実験開始当時に 3.6.0 も公開されていたが、LTS である 3.5.4 を選択した。

3. 検証項目及び実験概要

まず、純粋な鍵交換アルゴリズムの性能の測定として、鍵の生成、カプセル化・デカプセル化（X25519 の場合は公開鍵・共有秘密の生成）にかかる時間を調査した。

検証対象としたアルゴリズムは、X25519、ML-KEM 512、ML-KEM 768、ML-KEM 1024 である。尚、HQC は標準化対象に選定されたものの、FIPS 案が公開されていないことから今回は対象外とした。

各アルゴリズムについて、鍵の生成、カプセル化、デカプセル化それぞれについてかかる時間を 1000 回分計測し、オーバーヘッドの影響がある初回を除いた 999 回分のデータを基に算出した。

次に純粋な署名アルゴリズムの性能の測定として、鍵の生成、署名の生成・検証にかかる時間を調査した。

検証対象としたアルゴリズムは、2048-bit RSA、3072-bit RSA、P-256、ML-DSA 44 とした。尚、Falcon は標準化対象に選定されたものの 2026 年 1 月時点では FIPS の最終版が承認されていないことから、SLH-DSA は性能面で常用に堪えるものではないことから今回は対象外とした。

各アルゴリズムについて、鍵の生成、署名の生成、署名の検証それぞれについてかかる時間を 1000 回分計測し、オーバーヘッドの影響がある初回を除いた 999 回分のデータを基に算出した。

そして、実際の通信において、TLS ハンドシェイクの確立までにかかる時間を調査した。この通信は、二つの異なるアベイラビリティゾーンにある EC2 インスタンス間で、一方をクライアント、他方をサーバとして行う。このとき、サーバには基礎検証用に用意したサーバ証明書及び認証局証明書を提示させる。

検証対象とした鍵交換アルゴリズムは、X25519、ML-KEM 512、ML-KEM 768、ML-KEM 1024、X25519MLKEM768 である。また、サーバ及び認証局の署名アルゴリズムとして 2048-bit RSA、3072-bit RSA、P-256、ML-DSA 44 を使用している。尚、ここでいうサーバの署名アルゴリズムとは、Certificate Verify でサーバが署名に使用するア

ルゴリズムを指す。認証局の署名アルゴリズムとは、証明書に付されている署名アルゴリズムを指す。

ハンドシェイクの回数は 1000 回とし、その通信データを基に分析を行った。

後に提示する結果において、通信時間はクライアント及びサーバで記録された時間の差分によって算出している。尚、クライアントとサーバを動かしている EC2 インスタンスは同一の NTP サーバ(169.254.169.123)と同期させ、キャプチャファイルに記録された時間は NTP サーバとのずれを基に補正している。

また、参考までに、現在標準化候補として考察されている MAYO、CROSS、UOV、SNOVA をサーバの署名アルゴリズムとして採用した場合の結果も一部掲載する。

4. 検証結果

本文中の図は「図一覧」にてまとめて掲載している。

4-1. 鍵交換アルゴリズム

4-1-1. 性能の計測結果

まず、公開鍵・秘密鍵・暗号文・シークレットのサイズは以下の通りである:

	公開鍵	秘密鍵	暗号文	シークレット
X25519	32	32	32	32
ML-KEM 512	800	1632	768	32
ML-KEM 768	1184	2400	1088	32
ML-KEM 1024	1568	3168	1568	32

表 1:公開鍵・秘密鍵・暗号文・シークレットのサイズ(単位はバイト)

次に、純粋なアルゴリズムの性能について、鍵の生成にかかった時間の平均、標準偏差、各四分位及びロバストな変動係数[1]は以下の通りであった。尚、時間の分布については図 1 を参照のこと:

	平均	標準偏差	第一四分位	中央値	第三四分位	ロバストな変動係数
X25519	0.034287	0.002860	0.032659	0.033418	0.034522	0.055748
ML-KEM 512	0.010792	0.001629	0.010579	0.010670	0.010873	0.027554
ML-KEM 768	0.014644	0.002009	0.013911	0.014133	0.014519	0.043055
ML-KEM 1024	0.017122	0.001838	0.016500	0.016808	0.017057	0.033169

表 2:鍵交換アルゴリズムの鍵の生成にかかった時間の統計量(平均、標準偏差、各四分位の単位はミリ秒)

[1] 「ロバストな変動係数」は四分位範囲を中央値で割ったものとして算出している。これは分布に非対称性があることによる。

次に、カプセル化にかかった時間の平均、標準偏差、四分位及びロバストな変動係数は以下の通りであった。尚、時間の分布については図 2 を参照のこと：

	平均	標準偏差	第一四分位	中央値	第三四分位	ロバストな変動係数
X25519	0.064064	0.005161	0.061432	0.061660	0.064597	0.051322
ML-KEM 512	0.012838	0.002468	0.012215	0.012394	0.012560	0.027836
ML-KEM 768	0.015550	0.002440	0.014796	0.014929	0.015205	0.027396
ML-KEM 1024	0.017671	0.001962	0.017026	0.017260	0.017483	0.026506

表 3：カプセル化にかかった時間の統計量（平均、標準偏差、各四分位の単位はミリ秒）

最後に、デカプセル化にかかった時間の平均、標準偏差、四分位及びロバストな変動係数は以下の通りであった。尚、時間の分布については図 3 を参照のこと：

	平均	標準偏差	第一四分位	中央値	第三四分位	ロバストな変動係数
X25519	0.032776	0.003067	0.031598	0.031854	0.032543	0.029651
ML-KEM 512	0.010223	0.002208	0.009601	0.009748	0.010252	0.066680
ML-KEM 768	0.013716	0.002795	0.013071	0.013263	0.013540	0.035399
ML-KEM 1024	0.016848	0.001990	0.016273	0.016511	0.016726	0.027406

表 4：デカプセル化にかかった時間の統計量（平均、標準偏差、各四分位の単位はミリ秒）

4-1-2. 計測結果に対する評価

鍵生成・カプセル化・デカプセル化にかかった時間の平均は数十マイクロ秒のオーダーである。これに対し、TLS ハンドシェイクの確立にかかる時間はミリ秒のオーダーであることから、通常の通信用途では暗号処理にかかる時間の差は誤差の範囲であるといえる。

その上で敢えて評価をするならば、ML-KEM の鍵生成及びカプセル化にかかる時間は平均及びロバストな変動係数の結果から X25519 よりも高速かつ安定していると言える。一方、デカプセル化については、図 3 の分布の形状及びロバストな変動係数の結果から ML-KEM 512 では上振れしやすい傾向があると考えられる。

また、いずれのアルゴリズムも CPU 使用率やメモリ使用量等の計算資源の観点では負荷は極めて低く、今回使用した端末以上のスペックであれば暗号処理の負荷は問題にならないと考える。

ML-KEM の公開鍵・秘密鍵・暗号文のサイズは X25519 の 20 倍以上となる。

TLS ハンドシェイクの用途では、公開鍵は Client Hello で、暗号文は Server Hello でそれぞれ通信相手に送るため、公開鍵や暗号文が大きいならばそれだけ送信に必要なパケット数が増えることになる。

TCP の最大セグメントサイズは 1460 バイト(または 1440 バイト)がデファクトスタンダードであることから、ML-KEM 1024 では超過し、Client Hello 及び Server Hello で必要な TCP パケットが増加する。但し、Client Hello 及び Server Hello を送信するためには TCP パケットが 2 つあれば十分であった。

4-2. 署名アルゴリズム

4-2-1. 性能の計測結果

まず、鍵及び署名のサイズは以下の通りである。幅があるものについては、今回の基礎検証での実測値に依る:

	検証鍵	署名鍵	署名
2048-bit RSA	294	1213 - 1219	256
3072-bit RSA	422	1791 - 1795	384
P-256	91	138	69 - 72
ML-DSA 44	1312	2560	2420

表 5:検証鍵・署名鍵・署名のサイズ(単位はバイト)

次に、純粋なアルゴリズムの性能について、鍵の生成にかかった時間の平均、標準偏差、四分位及びロバストな変動係数は以下の通りであった。尚、時間の分布については図 4 及び図 5 を参照のこと:

	平均	標準偏差	第一四分位	中央値	第三四分位	ロバストな変動係数
2048-bit RSA	40.100951	24.470661	22.610428	34.956447	50.986659	0.811760
3072-bit RSA	141.447273	87.240103	75.312573	121.211471	185.203632	0.906606
P-256	0.035991	0.003434	0.034602	0.034802	0.035105	0.014453
ML-DSA 44	0.021171	0.002716	0.020317	0.020650	0.020974	0.031840

表 6:署名アルゴリズムの鍵の生成にかかった時間の統計量(平均、標準偏差、各四分位の単位はミリ秒)

次に、署名の生成にかかった時間の平均、標準偏差、四分位及びロバストな変動係数は以下の通りであった。尚、時間の分布については図 6 を参照のこと:

	平均	標準偏差	第一四分位	中央値	第三四分位	ロバストな変動係数
2048-bit RSA	0.284450	0.046486	0.263387	0.268292	0.285657	0.083008
3072-bit RSA	0.743106	0.094973	0.698908	0.703430	0.754942	0.079658
P-256	0.027631	0.003128	0.026358	0.026473	0.027143	0.029653
ML-DSA 44	0.056068	0.029422	0.035173	0.048201	0.069191	0.705732

表 7:署名の生成にかかった時間の統計量(平均、標準偏差、各四分位の単位はミリ秒)

最後に、検証にかかった時間の平均、標準偏差、四分位及びロバストな変動係数は以下の通りであった。尚、時間の分布については図 7 を参照のこと:

	平均	標準偏差	第一四分位	中央値	第三四分位	ロバストな変動係数
2048-bit RSA	0.037349	0.003374	0.035497	0.036482	0.037973	0.067869
3072-bit RSA	0.057842	0.004449	0.055404	0.055832	0.057474	0.037067
P-256	0.080259	0.005438	0.076950	0.078023	0.080582	0.046544
ML-DSA 44	0.021976	0.001927	0.021339	0.021473	0.021886	0.025451

表 8:署名の検証にかかった時間の統計量(平均、標準偏差、各四分位の単位はミリ秒)

4-2-2. 計測結果に対する評価

鍵生成にかかる時間について、RSA は平均もロバストな変動係数も他と比べるとかなり大きいことが分かる。即ち、生成に時間が掛かり、その時間も比較的幅があるといえる。但し、現時点では TLS ハンドシェイク毎に鍵を生成するといったことはないため、TLS 通信という観点ではそれほど問題にはならないと考えられる。

署名の生成や検証にかかる時間は、サーバ側の Certificate Verity の生成やクライアント側の Certificate/Certificate Verify の検証にかかる時間に影響を与える。

RSA での署名の生成にかかる時間は P-256 と比較して 10 倍以上、ML-DSA 44 と比較しても 5 倍以上となっている。尤も、1 ミリ秒未満であることから、よほど高速に通信を確立したいといった要望が無い限り、クライアントにとっては致命的といえるような遅さとはいえない。一方で、サーバにとっては署名アルゴリズムを変えることでわずかではあるが計算資源を節約できることになる。頻繁にハンドシェイクを行っている場合には考察の余地はある。

また、図 6 の分布の形状及びロバストな変動係数の結果から、ML-DSA 44 は署名の生成にかかる時間がやや上振れしやすいと考えられる。

署名の検証にかかる時間は P-256 が最も遅いが、数十マイクロ秒のオーダーであり、またロバストな変動係数もそこまで大きくないことから、通常の用途ではどのアルゴリズムを選択しても大差はないといえる。

その上で敢えて評価をするならば、ML-DSA 44 は平均及びロバストな変動係数の結果から従来の暗号と比べて高速かつ安定していると言える。

検証鍵はサーバ証明書に含まれ、署名は Certificate Verify で送信するため、これらのサイズが大きい場合にはそれだけサーバが送信する TCP パケットの数も増加することになる。

基礎検証においてサーバが Server Hello から Finished までを送信するために使用し

た TCP パケットの数について、結果を一部抜粋すると以下の通りであった：

認証局	2048-bit RSA			ML-DSA 44	
サーバ	2048-bit RSA	3072-bit RSA	ML-DSA 44	2048-bit RSA	ML-DSA 44
X25519	2	2	5	6	9
ML-KEM 512	3	3	5	7	9
ML-KEM 768	3	3	5	8	9
ML-KEM 1024	3	4	6	8	10
X25519MLKEM768	3	3	5	8	9

表 9：Server Hello から Finished までを送信するために使用した TCP パケットの数（抜粋）

尚、この値は署名アルゴリズムだけではなく証明書の記載の内容や中間証明書の有無にも左右されることは注記しておく。基礎検証において使用した証明書のサイズは「使用した証明書のサイズ」を参照のこと。

通信品質が理想的ではない場合は TCP 再送が発生し得る。TCP パケットの数が増えた場合、ハンドシェイク中に TCP 再送が発生する可能性は高くはなる。ハンドシェイク確立以降の通信が軽量であり、かつ可能な限りレイテンシを抑えたい場合には留意する必要があるだろう。加えて、輻輳ウィンドウの制限に留意する必要もある。

4-3. TLS ハンドシェイク

4-3-1. 計測結果

具体的な検証結果は別途配布している CSV を参照のこと。

4-3-2. 計測結果に対する評価及び仮説

ハンドシェイクの確立までにかかる時間を評価するにあたっていくつかの留意すべき事実がある。そのため、従来の署名アルゴリズムである 3072-bit RSA 及び P-256 と耐量子計算機暗号の ML-DSA 44 との比較は最後に述べる。

また、簡単のため、以降では計測の結果得られた平均時間を単に「時間」と書く。

4-3-2-1. 鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが 2048-bit RSA の場合

図 8 は、鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが 2048-bit RSA の場合に、サーバの署名アルゴリズムを変えた場合の各パートにかかる時間の結果である。

まず、UOV 及び SNOVA について、TCP ハンドシェイクの時点での遅れが目立っている。

TCP ハンドシェイクに対してその後に行われる TLS ハンドシェイクで使用する暗号の選択が影響を与えるとは考えられず、MAYO や CROSS では影響がないことから oqs-provider を利用した影響とも考えられない。サーバの署名アルゴリズムとして UOV と SNOVA を採用したケースは他のアルゴリズムと実験実施日が異なるため、それが通信品質に影響した可能性があり得る。

サーバの署名アルゴリズムとして RSA、P-256、ML-DSA 44、MAYO、CROSS を採用したケースの TCP ハンドシェイクは、SYN パケットの送信にかかる時間は 0.69 ミリ秒、ACK/SYN パケットの送信にかかる時間は 0.43 ミリ秒、ACK パケットの送信にかかる時間は 0.63 ミリ秒であった。従って、通信自体は往復で概ね 1 ミリ秒程度の時間がかかるといえる。

全体の傾向として、従来の RSA や P-256 と比較すると耐量子計算機暗号を使用した場合はハンドシェイクの確立にかかる時間が長くなることが分かる。

個別の傾向として、アルゴリズムによって以下の特徴が観測される：

特徴1. Server Hello を送信してから Finished の送信までの時間が殆どない（RSA、P-256、ML-DSA 44 等）

特徴2. Server Hello を送信してから Certificate を送信するまでの時間が目立つ（MAYO 2、SNOVA 25_8_3、UOV）

特徴3. Certificate を送信してから Certificate Verify を送信するまでの時間が目立つ（CROSS rsdp 128、SNOVA 37_17_2）

特徴4. Certificate Verify を送信してから Finished を送信するまでの時間が目立つ（CROSS rsdpg 128 fast）

特徴5. Client Hello を受信してから Server Hello を送信するまでの時間が短い（MAYO 2、UOV、SNOVA 37_17_2/25_8_3）

特徴6. Finished を受信してから Finished を送信するまでの時間が短い（RSA、CROSS rsdp 128 balanced/fast、SNOVA 37_17_2/25_8_3 等）

これら点について考察するが、その前に OpenSSL について確認する。

OpenSSL が送信データを送信バッファに書き込む際の挙動[2]として、まず、送信データのサイズが送信バッファの空き容量以下のときには送信バッファに溜めて終了する。

そうでない場合、送信バッファが空でなければまずは送信バッファに溜める。送信バッファを埋めたのち、送信バッファにあるデータを flush する。

（flush 後を含む）送信バッファが空の場合に、送信データが送信バッファサイズ以上であれば送信バッファを経由せずそのまま下位の BIO に書き込み続ける。下位の BIO への書き込

[2] openssl/crypto/bio/bf_buff.c 参照。

み後に送信バッファサイズ未満の送信データが残った場合または送信データが元々送信バッファサイズ未満の場合、送信バッファに溜めて終了する。

尚、下位の BIO への書き込みサイズは送信バッファサイズに制限されているわけではないため、送信データのサイズが送信バッファサイズの倍数でなかったとしても端数が送信バッファに残るとは限らない。また、送信バッファのサイズはデフォルトでは 4096 バイトとなっている。

この仕様を踏まえ、まず特徴 1 を考察する。特徴 1 は Server Hello から Finished までを殆ど同時に送信していることを意味する。特に、Server Hello の送信は Finished が送信可能になるまで遅らせたと考えられる。

OpenSSL の仕様から、サーバが送信可能なデータのサイズが 4096 バイトに到達するか、Finished の生成が完了するまで、メッセージは送信されない。従って、このケースではサーバが送信する各メッセージのサイズの合計が 4096 バイト以下であったと考えられる。

この点について、「4-3-2-1-2. 各メッセージの送信タイミングとサイズの関係」にて具体的に確認する。

特徴 2 について、まず Finished が生成される前に Server Hello が送信されていることから、送信バッファが埋まったと考えられる。本基礎検証においては Server Hello は最大でも 1663 バイトであったことから、Server Hello のみで送信バッファが埋まりきることはない。従って、flush された送信バッファ内のデータには Server Hello と少なくとも Certificate が含まれていると考えられる。

そして、Server Hello と Certificate の送信のタイミングが同時ではなかったという事実から、送信された Certificate は一部だけであり、残りは送信されなかったと考えられる。これについて、以下の二つが考えられる：

2-① Certificate の残りは送信バッファに溜められた

2-② Certificate の残りも送信するつもりだったが、別の理由で送信出来なかった

2-①の場合、OpenSSL の仕様から、残りの Certificate のサイズが 1 バイト以上 4096 バイト未満であった、即ち、Server Hello と Certificate のサイズの合計は 4096 バイトより大きく 8192 バイト未満であったと考えられる。

この点について、「4-3-2-1-2. 各メッセージの送信タイミングとサイズの関係」にて具体的に確認するが、図 8 における MAYO 2 及び SNOVA 25_8_3 では Server Hello と Certificate の合計サイズはそれぞれ 6788 バイトと 4196 バイトである。

一方、UOV は 2-②に当たると考えられる。この点について、「4-3-2-1-1. 各メッセージの送信タイミングと輻輳ウィンドウの関係」で述べる。

特徴 3 の場合も基本的には特徴 2 と同様で、送信バッファが埋まることで Certificate が送信されたと考えられる。このとき、Server Hello と Certificate が同時に送信された場合とされなかった場合の両方があり得る。

まず、Server Hello と Certificate が同時に送信された場合を考える。このとき、特徴 2 には該当しないことから、Server Hello と Certificate のサイズの合計は 4096 バイト以下または 8192 バイト以上であったと考えられる。

その上で、Certificate Verify は送信されていない必要がある。これも特徴 2 と同様に以下の二つが考えられる：

3-① Certificate Verify は送信バッファに溜められた

3-② Certificate Verify も送信するつもりであったが阻害された

まず、3-②については「4-3-2-1-1. 各メッセージの送信タイミングと輻輳ウィンドウの関係」で述べるが、CROSS rsdp 128 は 3-②に当たると考えられる。

次に、Server Hello と Certificate のサイズの合計が 4096 バイト以下の場合で 3-①の場合、Certificate Verify の一部が送信バッファに残り、Finished の生成を待つことになる。このケースでは、Certificate の送信から Certificate Verify の送信までの時間は 20 マイクロ秒から 30 マイクロ秒の範囲に収まっていた。Finished の生成自体が速いことが推察され、このケースではそれほど目立つ差にはならなかったと考えられる。

最後に、Server Hello と Certificate のサイズの合計が 8192 バイト以上の場合で 3-①の場合は、Certificate Verify は 4096 バイト未満であった。実際、図 8 における SNOVA 37_17_2 は Server Hello と Certificate のサイズの合計が 11718 バイトであり、Certificate Verify は 154 バイトである。

また、特徴 3 の場合で、Server Hello と Certificate が同時には送信されなかった場合、即ち特徴 2 に該当する場合は上記の 3-②に該当するケースに限られた。そのため、「4-3-2-1-1. 各メッセージの送信タイミングと輻輳ウィンドウの関係」にてまとめて述べる。

特徴 4 では、Finished を待たずに Certificate Verify 全てが送信されたと考えられる。一方、Finished は最後のメッセージであることから、生成され次第送信される。そして、特徴 3 の考察でも述べている通り、Finished の生成は数十マイクロ秒程度の時間で行われることが推察されている。そのため、特徴 4 がはっきり観測されるためには、Certificate Verify の送信後に Finished の送信がなんらかの理由で阻害される必要がある。

候補としては輻輳ウィンドウの制限と Nagle アルゴリズムであるが、OpenSSL はデフォルトでは TCP_NODELAY を有効とする[3]ため、輻輳ウィンドウが原因と考えられる。詳細は「4-3-2-1-1. 各メッセージの送信タイミングと輻輳ウィンドウの関係」で述べる。

[3] openssl/apps/lib/s_sockets.c 参照。

特徴 5 は特徴 1，特徴 2，特徴 3 の帰結と考えられる。

まず、Server Hello と Certificate の合計サイズが 4096 バイト以上の場合、Certificate の生成自体にはそれほど時間は掛からない[4]ため、実質的には Server Hello の生成が完了し次第送信したと考えられる。つまり、この送信タイミングが本来の Server Hello の送信タイミングと考えられる。

一方、Server Hello と Certificate の合計サイズが 4096 バイト未満の場合、後続のメッセージの生成を待ったために送信タイミングが後ろにずれていると考えられる。

これらから、アルゴリズムによって Client Hello を受信してから Server Hello を送信するまでの時間に形式的な記録上の差が出たと考えられる。

特徴 6 について、基本的には特徴 2，特徴 3，特徴 4 による各メッセージの送信タイミングの違いによって説明出来る。

まず、クライアントがサーバからの Finished を受信したとき、メッセージ認証コードを検証した後、同じように Finished を生成する。しかし、各メッセージの受信のタイミングによっては、Finished を受信しても Certificate Verify 以前のメッセージを処理している場合がある。

特徴 2，特徴 3，特徴 4 によって各メッセージの送信タイミングがずれた場合、クライアントの Finished の受信と Certificate Verify 以前のメッセージの受信に差が生じうる。このとき、クライアントが Finished を受信した時点では Certificate Verify 以前のメッセージの処理に既に着手しているため、記録上は短縮されたと考えられる。

但し、RSA については例外的な事情が示唆される。この点は「4-3-2-2-1. サーバと認証局で同じ署名アルゴリズムを使用した場合」にて確認する。

4-3-2-1-1. 各メッセージの送信タイミングと輻輳ウィンドウの関係

まず、TCP 通信の輻輳制御の一つである輻輳ウィンドウの仕様について確認する。

RFC 5681[5]では、TCP 通信において送信者が一方的に送信できるデータ量として輻輳ウィンドウを定め、新しいデータに対する ACK パケットを受信するたびに輻輳ウィンドウを増加させることとしている。そして、RFC 6928[6]では輻輳ウィンドウの初期値を 10MSS とし、TCP ハンドシェイク中の ACK パケット、SYN/ACK パケットでは増加させるべきではないとしている。また、RFC 7661[7]の Section 4.4 では、送信者が輻輳ウィンドウの半分も使っていない場合、輻輳ウィンドウを更新しないこととしている。

---

[4] Certificate は既に登録されているサーバ証明書と認証局証明書を送信するだけのため。
[5] Ethan Blanton, Dr. Vern Paxson, and Mark Allman, “TCP Congestion Control”, RFC 5681, September 2009
[6] Jerry Chu, Nandita Dukkipati, Yuchung Cheng, and Matt Mathis, “Increasing TCP’s Initial Window”, RFC 6928, April 2013
[7] Gorry Fairhurst, Arjuna Sathiaseelan, and Raffaello Secchi, “Updating TCP to Support Rate-Limited Traffic”, RFC 7661, October 2015

TLS ハンドシェイクにおいては、サーバが最初に送る新しいデータとは Server Hello となるため、Server Hello を送信する時点ではサーバは最大で 10MSS までしか送信できないことになる。尚、OS の実装上、バイト数ではなくセグメント数で管理している場合がある[8]。この場合、送信データのサイズとしては 10MSS に到達していなくとも 10 パケット分送信すると送信が制限される。

尚、今回の基礎検証は IPv4 での通信であり、タイムスタンプオプションが有効となっているため、1MSS は 1448 バイトとなっている。

この仕様を踏まえ、前節における挙動 2-②を考察する。

まず、OpenSSL の仕様に基づくと、Server Hello と Certificate のサイズの合計は 8192 バイト以上であれば、Server Hello と Certificate の一部を送信した後に Certificate の残りも送信されると考えられる。

しかし、Server Hello と Certificate を送るために 11 パケット以上必要な場合、Certificate の送信中に輻輳ウィンドウの制限を受け、クライアントからの ACK パケットを待たなければならなくなる。

Server Hello と Certificate のサイズの合計が 8192 バイト以上の場合、最初に flush される 4096 バイトで 3 パケット使用し、残りの Certificate を 1448 バイトごとに分割して送信する。

従って、Server Hello と Certificate のサイズの合計が 14232 バイトを超える場合は Certificate の送信中に輻輳ウィンドウの制限を受け、Server Hello を送信してから Certificate を送信するまでに時間差が生ずると考えられる。

この点について、「4-3-2-1-2. 各メッセージの送信タイミングとサイズの関係」にて確認する。

同様に、Certificate Verify の送信中に輻輳ウィンドウの制限を受けることも考えられる（前節の 3-②）。Certificate Verify が送信されているという状況は、OpenSSL の仕様を踏まえると、まず以下の三つに分けられる：

① Server Hello と Certificate のサイズの合計が 4096 バイト未満
② Server Hello と Certificate のサイズの合計が 4096 バイト以上 8192 バイト未満
③ Server Hello と Certificate のサイズの合計が 8192 バイト以上 12784 バイト以下

③の 12784 バイトは Server Hello と Certificate が 9 パケット以内に収まることを意味する。

上記の状況で輻輳ウィンドウの制限を受けるためには、それぞれ対応する以下の条件が必

[8] linux/net/ipv4/tcp_cubic.c 参照。

要となる：

条件①　Server Hello から Certificate Verify までのサイズの合計が 14232 バイトを超える

条件②　Server Hello から Certificate Verify までのサイズの合計が 13984 バイトを超える

条件③　Certificate Verify のサイズが送信可能なパケット数に 1448 バイトを掛けた値を超える

尚、これらの条件を満たす組み合わせは、全て署名アルゴリズムとして CROSS を採用していた。

条件①を満たすとき、Certificate の送信から Certificate Verify の送信までにかかる時間は 1.03 ミリ秒以上であった。条件③を満たすとき、0.98 ミリ秒以上であった。従って、これらの場合は輻輳ウィンドウの制限を受けて Certificate Verify の送信が阻害されたと考えられる。

一方、条件②を満たす場合であっても Certificate の送信から Certificate Verify の送信までにかかる時間の大半が 0.60 ミリ秒以下であり、中には 0.10 ミリ秒を切る組み合わせもある。

条件②を満たすとき、Certificate の送信は Certificate Verify の生成を待つ。そのため、Server Hello の送信から Certificate Verify の送信までに時間差が生じ、その間に最初に送信した 4096 バイトに対するクライアントからの ACK パケットを受信することがある。これにより輻輳ウィンドウがスライドし、Certificate の送信から Certificate Verify の送信までの間に輻輳ウィンドウの制限を受けにくくなったと考えられる。

尚、Server Hello の送信から Certificate Verify の送信までにかかる時間でみると、条件①を満たすとき 1.04 ミリ秒以上、条件②を満たすとき、0.99 ミリ秒以上、条件③を満たすとき 1.01 ミリ秒以上となった。

最後に、Finished の送信が輻輳ウィンドウの制限を受ける場合を考察する。これについて、以下の二つの状況が考えられる：

① Certificate Verify は Finished の生成を待たずに送信され、Certificate Verify までででちょうど輻輳ウィンドウの上限に達した

② Certificate Verify は Finished と同時に送信されたが、Finished が分割され、Finished の前半までででちょうど輻輳ウィンドウの上限に達した

ACK パケットの受信によって輻輳ウィンドウのスライドや増加が生じるため、各メッセージのサイズだけでは判断することが出来ない。そこで、実際に Finished の送信が輻輳ウィンドウの制限を受けたと考えられる場合を個別に確認する。

まず、Server Hello から Finished までを送信するために必要なパケットが 10 以下である場合、輻輳ウィンドウの初期値内であることから輻輳ウィンドウの制限は受けない。このと

きの Certificate Verify の送信から Finished の送信までにかかる時間は 0.04 ミリ秒以内であった。従って、0.04 ミリ秒以上かかった組み合わせでは Finished の送信が輻輳ウィンドウの制限を受けた可能性が示唆される。

次に、今回の基礎検証中に Finished が分割された組み合わせでは Server Hello から Finished までを送信するために必要なパケットが 7 であったことから、Finished の後半が輻輳ウィンドウの制限によって送信出来なかったケースは存在しなかった。

Certificate Verify の送信から Finished の送信までにかかる時間が 0.04 ミリ秒以上かかった組み合わせは、0.12 ミリ秒のもの、0.57 ミリ秒のもの、1.01 ミリ秒以上かかったものに分けられる。Server Hello から Certificate Verify までを送信するために必要なパケット数は、それぞれ 16 パケット、13 パケット、10 パケットであった。

逆に、Server Hello から Certificate Verify までを送信するために必要なパケット数が 10/13/16 パケットであったとしても、必ずしも Finished が輻輳ウィンドウの制限にかかるわけではなかった。以下、この違いについて考察する。

まず、Server Hello から Certificate Verify までを送信するために必要なパケット数が 10 パケットの場合である。この場合に Finished が輻輳ウィンドウの制限にかからなかった組み合わせでは、Server Hello と Certificate の合計が 4096 バイト以上であった。一方、Finished が輻輳ウィンドウの制限にかかった場合は 4096 バイト未満である。

Server Hello と Certificate の合計が 4096 バイト以上の場合、Server Hello と Certificate の一部が flush されることから、Finished を送信するまでにクライアントからの ACK パケットを受信する可能性がある。そのため、Finished を送信する時点では輻輳ウィンドウに余裕があると考えられる。一方、4096 バイト未満の場合は Server Hello から Certificate Verify までが殆ど同時に送信されることから ACK パケットを受信する機会がなく、Server Hello から Certificate Verify までで 10 パケット使ってしまうことから、Finished が輻輳ウィンドウの制限にかかってしまうと考えられる。

次に、Server Hello から Certificate Verify までを送信するために必要なパケット数が 13 パケットの場合である。この場合も同様に、Server Hello と Certificate の合計が 4096 バイト以上となる組み合わせでは Finished は輻輳ウィンドウの制限にかからなかった。

しかし、4096 バイト未満では輻輳ウィンドウの制限にかかる場合とそうでない場合があった。鍵交換アルゴリズムが ML-KEM512、認証局の署名アルゴリズムが ML-DSA 44 のときであり、サーバの署名アルゴリズムが CROSS rsdpg 128 balanced では制限にかからず、CROSS rsdpg 128 small では制限にかかった。

この二つの違いは署名の生成にかかる時間の違いであり、CROSS rsdpg 128 small は CROSS rsdpg 128 balanced よりも 0.33 ミリ秒遅い。即ち、CROSS rsdpg 128

small の場合は Certificate Verify が送信可能となるタイミングが相対的に遅くなる。その結果、Certificate Verify が送信可能となる前に、最初に送信した 4096 バイトに対するクライアントからの ACK パケットを受信する可能性が高くなる。

クライアントからの ACK パケットを受信したとき、既に送信したパケット数が輻輳ウィンドウの半分に満たないことから、輻輳ウィンドウは更新されない。その結果、10 パケットしか送ることが出来ず、Certificate の残りと Certificate Verify で到達してする。従って、Finished が輻輳ウィンドウの制限にかかったと考えられる。

一方、CROSS rsdpg 128 balanced は送信可能となるタイミングが相対的に早くなることで、クライアントからの ACK パケットを受信する前に Certificate Verify が送信可能となる場合がある。そのため、Certificate Verify の送信中に輻輳ウィンドウの制限にかかりやすく、Finished が送信可能となるときには輻輳ウィンドウがスライドまたは増加して制限がかかりにくくなったと考えられる。

最後に、Server Hello から Certificate Verify までを送信するために必要なパケット数が 16 パケットの場合である。この場合は逆に Server Hello と Certificate の合計が 4096 バイト未満となる組み合わせでは Finished は輻輳ウィンドウの制限にかからなかった。この組み合わせの場合、Certificate Verify の送信中に輻輳ウィンドウの制限にかかるため、Finished が送信可能となるときには輻輳ウィンドウがスライドまたは増加して制限がかかりにくくなったと考えられる。

一方、Server Hello と Certificate の合計が 4096 バイト以上となる組み合わせの場合、Certificate Verify の送信中にクライアントからの ACK パケットを受信する可能性がある。ACK パケット受信時点で送信パケット数が輻輳ウィンドウの半分以上となっている場合、輻輳ウィンドウが増加して 13 となる。この 13 は Certificate の残りと Certificate verify で到達してしまうため、Finished が輻輳ウィンドウの制限にかかったと考えられる。

4-3-2-1-2. 各メッセージの送信タイミングとサイズの関係

まず、図 9 及び図 10 はサーバが送信する各メッセージのサイズの合計と Server Hello の送信から Finished の送信までにかかる時間の関係を示している。

各メッセージのサイズの合計が 4096 バイト以下であるときは Server Hello の送信から Finished の送信までにかかる時間が 0.005 ミリ秒以下であったのに対し、4096 バイトを超えると 0.02 ミリ秒を超えている。

この差から、各メッセージのサイズの合計が 4096 バイト以下となるようなアルゴリズムの組み合わせの場合、Server Hello の送信は Finished が送信可能になるまで待たされていると考えられる。

次に、図 11 及び図 12 は Server Hello と Certificate のサイズの合計と Server

Hello の送信から Certificate の送信までにかかる時間の関係を示している。
　Server Hello と Certificate のサイズが 4096 バイト未満の場合、Server Hello の送信から Certificate の送信までにかかる時間は 0.005 ミリ秒以下である。このことから、この二つのメッセージが送信可能となるタイミングは殆ど同時となると考えられる。

　Server Hello と Certificate のサイズが 4096 バイト以上かつ 8192 バイト未満となる場合、Server Hello の送信から Certificate の送信までにかかる時間は、一部の組み合わせ以外では 0.44 ミリ秒以上であることから、Certificate の送信は Certificate Verify が送信可能となるまで待たされたと考えられる。
　尚、一部の組み合わせとはサーバの署名アルゴリズムとして P-256 を採用した場合である。このとき、Server Hello の送信から Certificate の送信までにかかる時間は 0.10 ミリ秒であった。このことから、サーバの署名アルゴリズムとして P-256 を採用した場合は Certificate Verify の生成が高速であることも示唆される。
　また、図 11 では 0.90 ミリ秒から 1.50 ミリ秒掛かっているアルゴリズムもみられる。これらに該当するのは、サーバの署名アルゴリズムが 3072-bit RSA、CROSS rsdp 128 balanced/small、CROSS rsdpg 128 small のときである。これらのアルゴリズムは署名の生成が比較的遅いため、それが反映されていると考えられる。

　Server Hello と Certificate のサイズが 8192 バイト以上であり、かつ 14232 バイト以下である場合、Server Hello の送信から Certificate の送信までにかかる時間は 0.032 ミリ秒以下であった。比較的時間が掛かっているものの、これらの場合で Certificate の送信から Certificate Verify の送信までにかかる時間は、一部の組み合わせ以外では 0.41 ミリ秒以上であることから、Certificate は Certificate Verify が送信可能となる前に送信されたと考えられる。
　尚、一部の組み合わせとはサーバの署名アルゴリズムとして P-256 を採用した場合である。このとき、Certificate の送信から Certificate Verify の送信までにかかる時間は 0.07 ミリ秒から 0.08 ミリ秒であったことから、この場合でもサーバの署名アルゴリズムとして P-256 を採用した場合は Certificate Verify の生成が高速であることも示唆される。

　最後に、Server Hello と Certificate のサイズが 14232 バイトを超える場合、Server Hello の送信から Certificate の送信までにかかる時間は最低でも 1.44 ミリ秒であった。この時間差は TCP 通信の往復一回分以上であることからも、クライアントからの ACK パケットを待っていたことが示唆される。

4-3-2-2. 鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが P-256 または ML-DSA 44 の場合

まず、図 13 は鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが P-256 の場合にサーバの署名アルゴリズムを変えた場合の各パートにかかる時間の結果である。次に、図 14 は鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが ML-DSA 44 の場合にサーバの署名アルゴリズムを変えた場合の各パートにかかる時間の結果である。

認証局の署名アルゴリズムが 2048-bit RSA の場合(図 8)と図 13、図 14 とを比較すると、ハンドシェイクの確立までにかかる時間について以下の特徴が確認される:

1. RSA、P-256、ML-DSA 44 は、サーバと認証局で同じ署名アルゴリズムを使用した場合、異なる署名アルゴリズムを使用した場合と比べて高速化する
2. MAYO 1 や SNOVA(37_17_2 を除く)は、認証局の署名アルゴリズムが ML-DSA 44 の方が高速化している

以下では 1.について考察する。2.については「4-3-2-4. Certificate の受信のタイミングによる高速化」で述べる。

4-3-2-2-1. サーバと認証局で同じ署名アルゴリズムを使用した場合

図 8、図 13、図 14 の RSA、P-256、ML-DSA 44 では、Finished の受信から Finished の送信までにかかる時間に違いがみられる。Finished の受信から Finished の送信までにかかる時間にはそれ以前に受信したメッセージの処理時間が含まれる可能性があることから、クライアントの処理時間として Server Hello の受信から Finished の送信までの時間で比較をする。

まず、鍵交換アルゴリズムが X25519 の場合、以下の通りであった:

		認証局			
		2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
サーバ	2048-bit RSA	0.572286	0.589398	1.330819	1.707963
	3072-bit RSA	0.611546	0.639316	1.400812	2.369867
	P-256	1.310253	1.332677	0.569139	1.432059
	ML-DSA 44	1.447609	1.464957	1.425592	0.984149

表 10:鍵交換アルゴリズムが X25519 の場合の Server Hello の受信から Finished の送信までの時間(単位:ミリ秒)

次に、ML-KEM 512 では以下の通りであった：

		認証局			
		2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
サーバ	2048-bit RSA	0.620518	0.636354	1.370692	1.508257
	3072-bit RSA	0.660574	0.682039	1.417594	1.534633
	P-256	1.364056	1.386600	0.608567	1.490710
	ML-DSA 44	1.493633	1.876917	1.478147	1.017371

表 11：鍵交換アルゴリズムが ML-KEM 512 の場合の Server Hello の受信から Finished の送信までの時間（単位：ミリ秒）

最後に、X25519MLKEM768 では以下の通りであった：

		認証局			
		2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
サーバ	2048-bit RSA	0.706700	0.730319	1.459543	1.596950
	3072-bit RSA	0.751469	0.778619	1.499156	1.624995
	P-256	1.450575	1.475592	0.698734	1.569820
	ML-DSA 44	1.850925	1.830818	1.548543	1.064178

表 12：鍵交換アルゴリズムが X25519MLKEM768 の場合の Server Hello の受信から Finished の送信までの時間（単位：ミリ秒）

表 10、表 11、表 12 で色が付けているセルは、Server Hello と Certificate の合計サイズが 4096 バイト以上 8192 バイト未満となる組み合わせである。この組み合わせでは Server Hello と Certificate の一部が先に送信され、Certificate の残りは Certificate Verify 以降のメッセージを待つことになる。

この場合、クライアントは Server Hello を受信して処理した後に Certificate を待つ時間が発生するため、他の項目と比べてアルゴリズムの差以上に大きくなりやすいことに留意する必要がある。

また、表 10、表 11、表 12 から、サーバと認証局で同じ署名アルゴリズムを使用した場合、他の組み合わせと比べるとクライアントの暗号処理の時間が短くなる傾向がみられる。以下、その理由について考察する。

サーバと認証局で同じ署名アルゴリズムを使用する場合、サーバ証明書につけられた署名、認証局証明書につけられた署名、Certificate Verify で送信される署名が全て同じアルゴリズムとなる。このとき、クライアントは Certificate Verify に含まれる署名の検証では、Certificate の証明書の検証の際に読み込んだ命令をそのまま再利用することが出来る。その結果、署名の検証のためのオーバーヘッドが軽減され、Finished を生成・送信するまでの

時間が短くなると考えられる。
　従って、ある証明書の署名アルゴリズムを変更した場合の性能面での影響を測定する際には、中間認証局証明書や証明書内の各項目で使用する署名アルゴリズムの影響を考慮する必要がある。

4-3-2-3. 鍵交換アルゴリズムの選択による変化

　図 15 は、サーバ及び認証局の署名アルゴリズムが 2048-bit RSA の場合に、鍵交換アルゴリズムを変えた場合の各パートにかかる時間の結果である。
　ハンドシェイク全体にかかる時間は、X25519 では 3.69 ミリ秒、ML-KEM 512 では 3.72 ミリ秒と極端な差は見られない。一方、ハイブリッドである X25519MLKEM768 では 3.96 ミリ秒となっており、X25519 と比較した場合には 0.27 ミリ秒の差である。
　この差は主に、ACK の送信から Client Hello の送信までの 0.06 ミリ秒、Client Hello の受信から Server Hello の送信までの 0.06 ミリ秒、Finished の受信から Finished の送信までの 0.13 ミリ秒に由来する。
　尚、通常の用途ではそこまで大きな差であるとは言えない。

　図 16 は、サーバ及び認証局の署名アルゴリズムが ML-DSA 44 の場合に、鍵交換アルゴリズムを変えた場合の各パートにかかる時間の結果である。
　鍵交換も署名も格子暗号で揃えたとしても、そのことによる高速化は確認されなかった。

　図 15 及び図 16 では鍵交換アルゴリズムが X25519MLKEM768 の場合に最も時間が掛かっている。このことは、鍵長が長いと送信するデータが増えるために処理時間が掛かる、あるいはハイブリッドで複雑になるほど暗号処理のオーバーヘッドで時間が掛かるという自然な結果である。
　しかし、署名アルゴリズムと鍵交換アルゴリズムの組み合わせによっては鍵や署名のサイズが大きい方が却って高速化する事例が確認されている。次節ではこの事例を考察する。

4-3-2-4. Certificate の受信のタイミングによる高速化

　図 17 は、鍵交換アルゴリズムが X25519、サーバの署名アルゴリズムが SNOVA 24_5_4 の場合に、認証局の署名アルゴリズムを変えた場合の各パートにかかる時間の結果である。

　図 17 では Certificate で送信される証明書のサイズが最も大きくなる ML-DSA 44 の場合に最も早くハンドシェイクを確立させている。ハンドシェイク全体にかかる時間は、ML-DSA 44 のときは 5.51 ミリ秒であり、次に早い 2048-bit RSA のときで 5.97 ミリ秒と

なっている。ML-DSA 44 に変更することによって 0.46 ミリ秒の短縮となっている。
ML-DSA 44 とそれ以外とで異なるのは、Server Hello（及び Certificate）の送信のタイミングである。クライアントが Client Hello を送信してから Server Hello を受信するまでの時間は、ML-DSA 44 のときは 1.98 ミリ秒であり、次に早い 2048-bit RSA で 2.57 ミリ秒となっている。
これに対し、暗号処理にかかる時間にはそれほど差はない。実際、サーバの目線では、Client Hello の受信から Finished の送信までの時間は、認証局の署名アルゴリズムが 2048-bit RSA のときは 1.22 ミリ秒、3072-bit RSA のときは 1.23 ミリ秒、P-256 のときは 1.26 ミリ秒、ML-DSA 44 のときは 1.32 ミリ秒となっている。
同様に、クライアントの目線では、Server Hello の受信から Finished の送信までの時間は、認証局の署名アルゴリズムが 2048-bit RSA のときは 1.77 ミリ秒、3072-bit RSA のときは 1.81 ミリ秒、P-256 のときは 1.79 ミリ秒、ML-DSA 44 のときは 1.88 ミリ秒となっている。

また、図 18 は、サーバの署名アルゴリズムが CROSS rsdp 128 balanced、認証局の署名アルゴリズムが ML-DSA 44 の場合に、鍵交換アルゴリズムを変えた場合の各パートにかかる時間の結果である。

図 18 では、鍵のサイズが大きくなった方が寧ろ早くハンドシェイクを確立させている。ハンドシェイク全体にかかる時間は、ML-KEM 768 のときは 5.29 ミリ秒、X25519MLKEM768 のときは 5.50 ミリ秒、X25519 のときは 6.02 ミリ秒となっている。
図 17 と違い、クライアントの Client Hello の送信から Server Hello の受信までの時間ではそれほど差はない。最も短いのは ML-KEM 768 のときの 1.83 ミリ秒、最も長いのは X25519MLKEM768 のときの 1.91 ミリ秒である。
一方、Certificate の受信までを含めると、X25519MLKEM768 では 1.91 ミリ秒であるのに対し、X25519 では 2.82 ミリ秒となっている。
また、クライアントが Certificate を受信してから Finished を送信するまでの時間は、X25519 及び ML-KEM 512 では 1.91 ミリ秒、ML-KEM 768 では 2.13 ミリ秒、ML-KEM 1024 では 2.14 ミリ秒、X25519MLKEM768 では 2.21 ミリ秒である。Server Hello の受信のタイミングの問題で X25519 及び ML-KEM 512 では Server Hello の処理時間が含まれていない可能性があることを考慮すれば、暗号処理の時間にそれほど差があるとは言えない。

Certificate の受信のタイミングが早期化することでクライアントは早期に暗号処理に着手しつつ、並行して Certificate Verify 以降のメッセージを待つことが出来るため、ハンドシ

ェイク全体が高速化したと考えられる。このことは、鍵交換アルゴリズムが X25519、サーバの署名アルゴリズムが SNOVA 24_5_4 の場合でも整合的である。

4-3-2-5. 3072-bit RSA/P-256/ML-DSA 44 の比較

2048-bit RSA の安全性は 112 ビットセキュリティであることから、耐量子性とは無関係に 2030 年までにより安全な移行する必要がある。その際の移行先として、128 ビットセキュリティの安全性を持つ 3072-bit RSA、P-256、ML-DSA 44（あるいは同等以上の安全性を持つ他の各種 ECDSA）が検討される。

純粋なアルゴリズムの性能については既に述べた通りであり、少なくとも本基礎検証と同等以上の計算資源がある場合にはどれを選択しても明確に差があるとは言えない。

ハンドシェイクの確立にかかる時間という観点では、前節までで考察した通り、アルゴリズムの組み合わせによって高速化し得る。そのため、一概には言えない。

そこで、以下の二つの観点で暫定的な評価を与える：

1. 一つの署名アルゴリズムを認証局の署名アルゴリズムとし、残りの二つをサーバの署名アルゴリズムとして採用した場合を比較する
2. サーバと認証局で同一の署名アルゴリズムを採用した場合を比較する。

この二つの観点のいずれの場合においても、ハンドシェイクの確立までにかかる時間の平均が最も短いのは P-256、次いで ML-DSA 44 で、最も長いのは 3072-bit RSA であった。従って、理想的な通信環境であれば、ML-DSA 44 は 3072-bit RSA よりも高速にハンドシェイクを確立できると言える。

但し、シークレットの暗号文や署名のサイズが大きくなることで、サーバが送信するパケット数が増えることに留意する必要がある。インターネットを経由する場合にはパケットロスの可能性もあることから、ML-DSA 44 と 3072-bit RSA との差の 1 ミリ秒はパケットロス率によっては逆転しうる。加えて、サーバ証明書や中間認証局証明書のサイズが大きくなることで輻輳ウィンドウの制限にかかる可能性がある。

図一覧

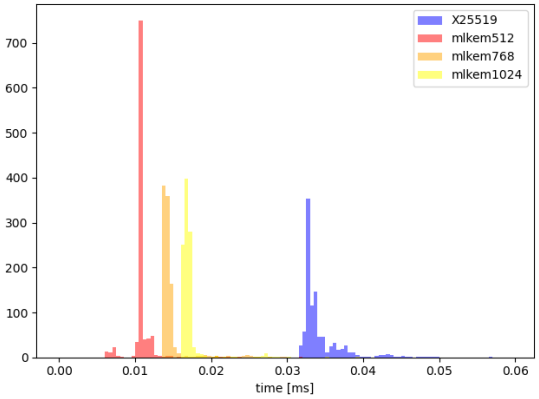

図 1:鍵交換アルゴリズムの鍵の生成にかかった時間の分布

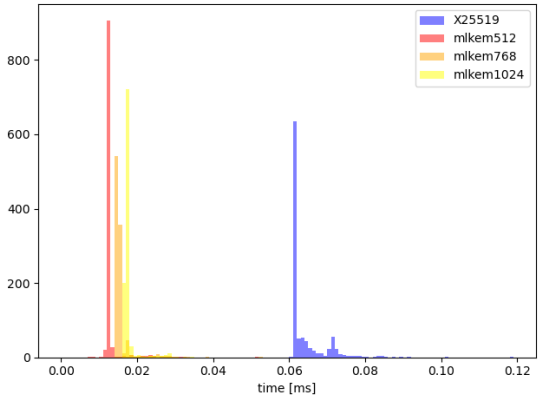

図 2:カプセル化にかかった時間の分布

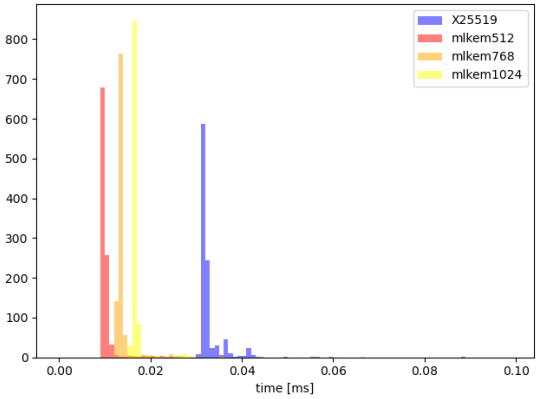

図 3:デカプセル化にかかった時間の分布

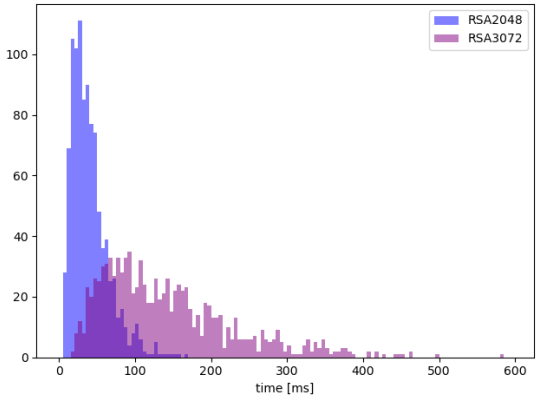

図 4:署名アルゴリズムの鍵生成にかかった時間の分布（RSA）

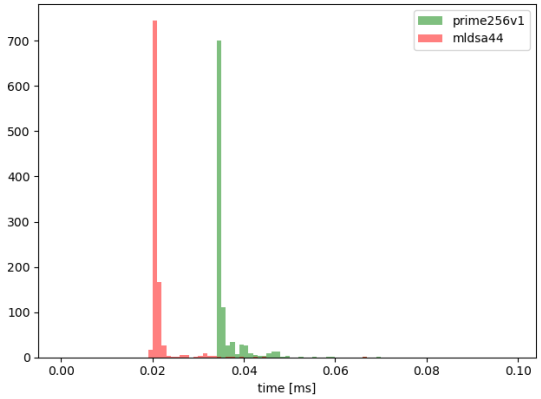

図 5:署名アルゴリズムの鍵生成にかかった時間の分布（P-256、ML-DSA 44）

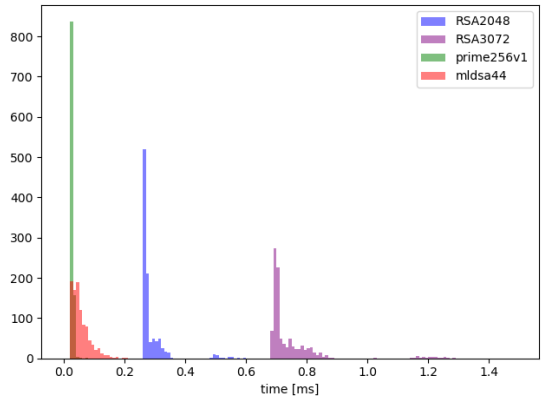

図 6:署名の生成にかかった時間の分布

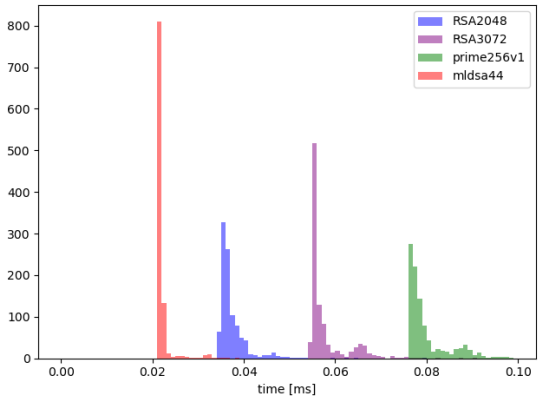

図 7:署名の検証にかかった時間の分布

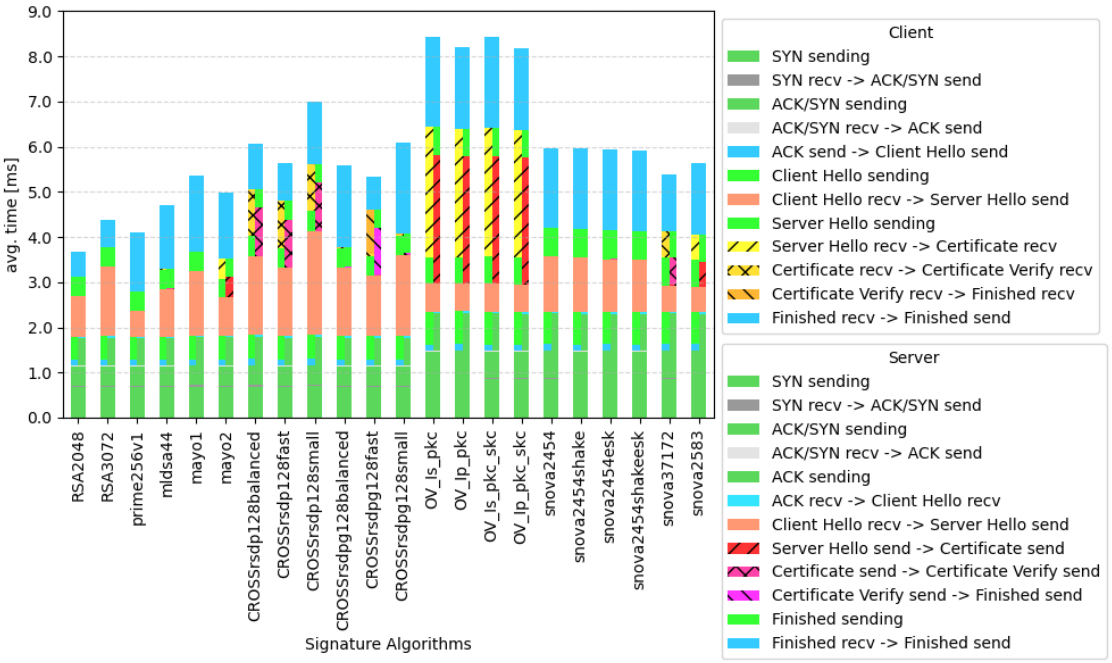

図 8:鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが 2048-bit RSA の場合

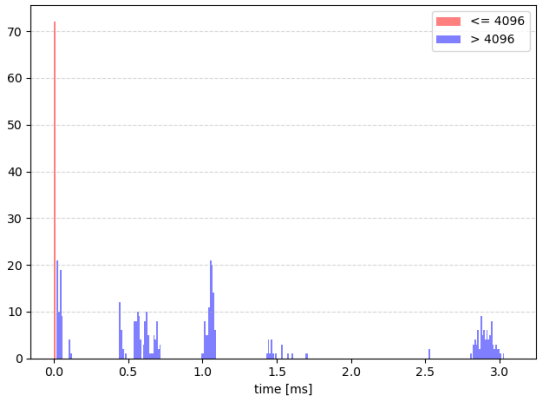

図 9:サーバが送信する各メッセージのサイズの合計と Server Hello の送信から Finished の送信までにかかる時間の関係

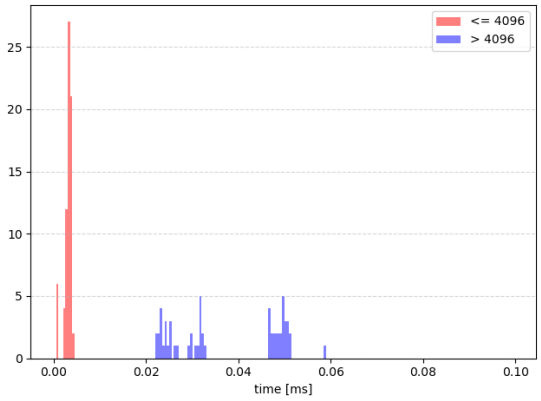

図 10:図 9 を 0.0 ミリ秒から 0.1 ミリ秒の範囲に制限したもの

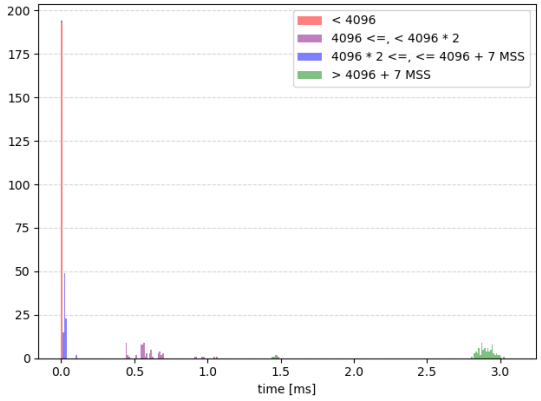

図 11:Server Hello と Certificate のサイズの合計と Server Hello の送信から Certificate の送信までにかかる時間の関係

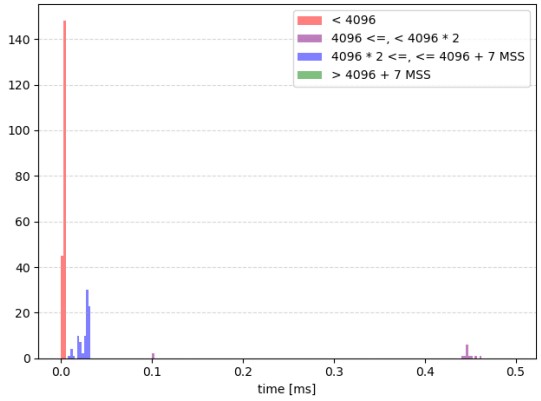

図 12:図 11 を 0.0 ミリ秒から 0.5 ミリ秒の範囲に制限したもの

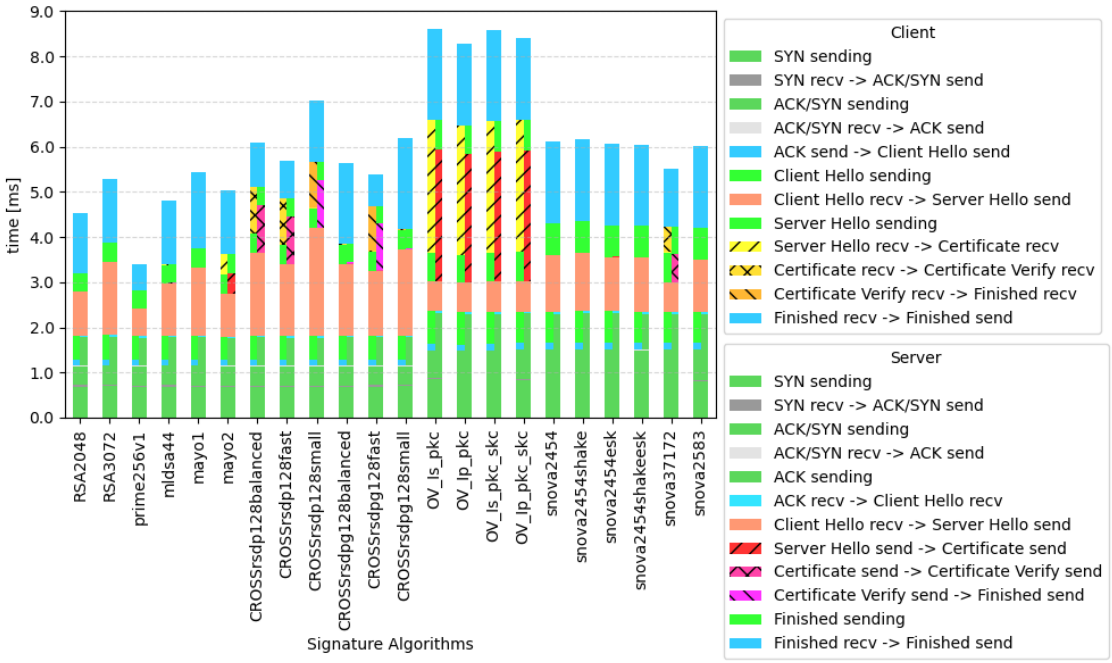

図 13：鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが P-256 の場合

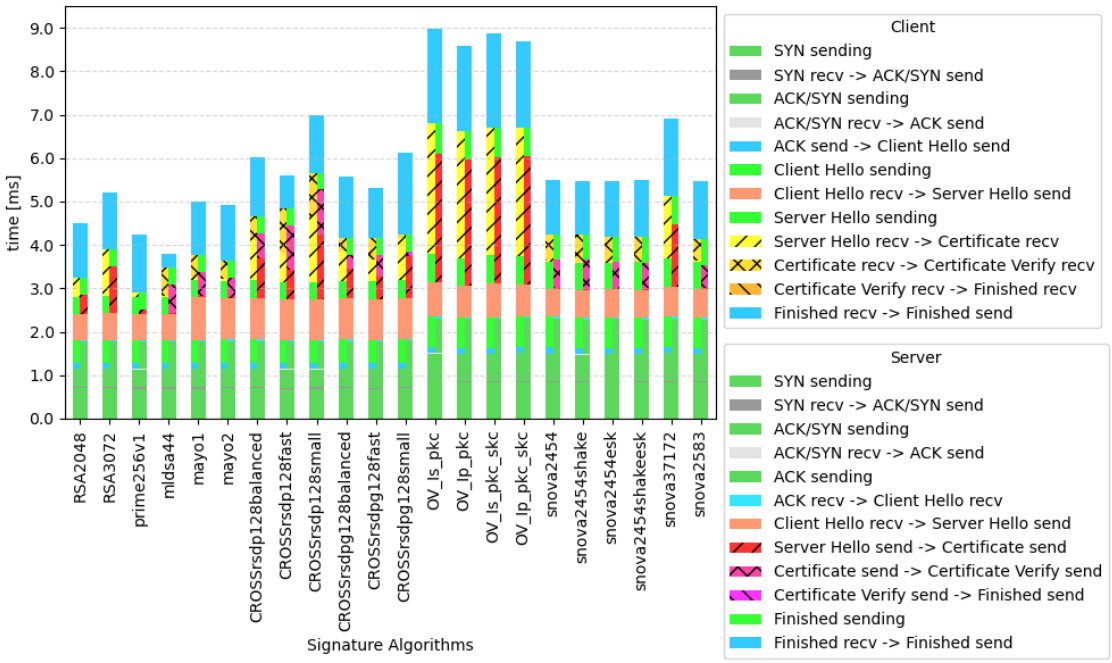

図 14：鍵交換アルゴリズムが X25519、認証局の署名アルゴリズムが ML-DSA 44 の場合

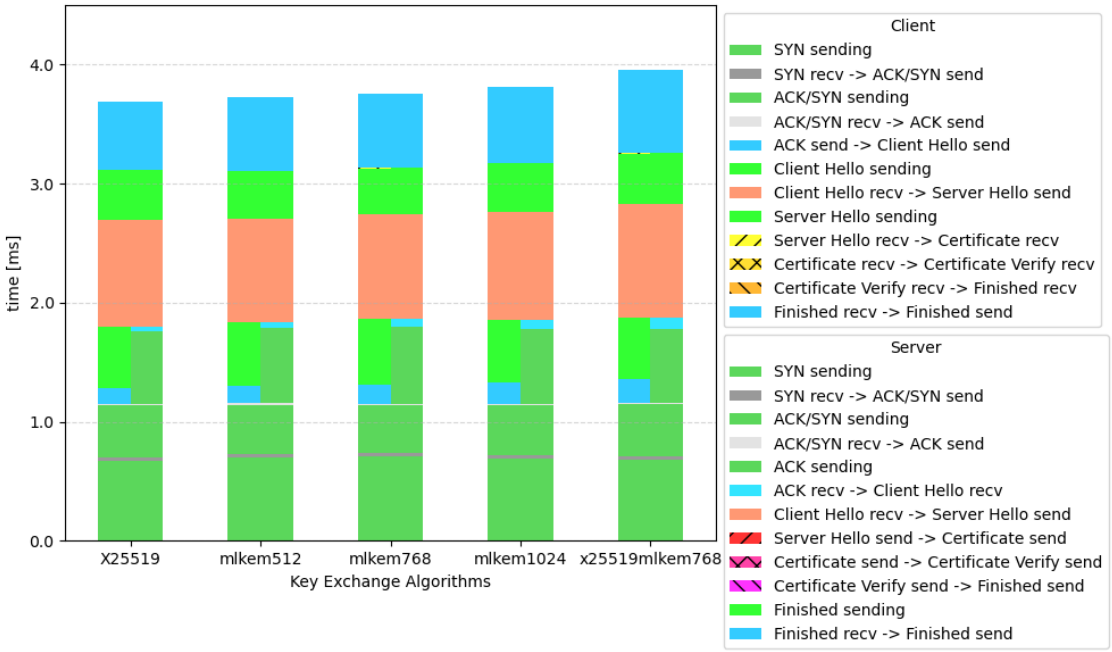

図 15：サーバ及び認証局の署名アルゴリズムが 2048-bit RSA の場合

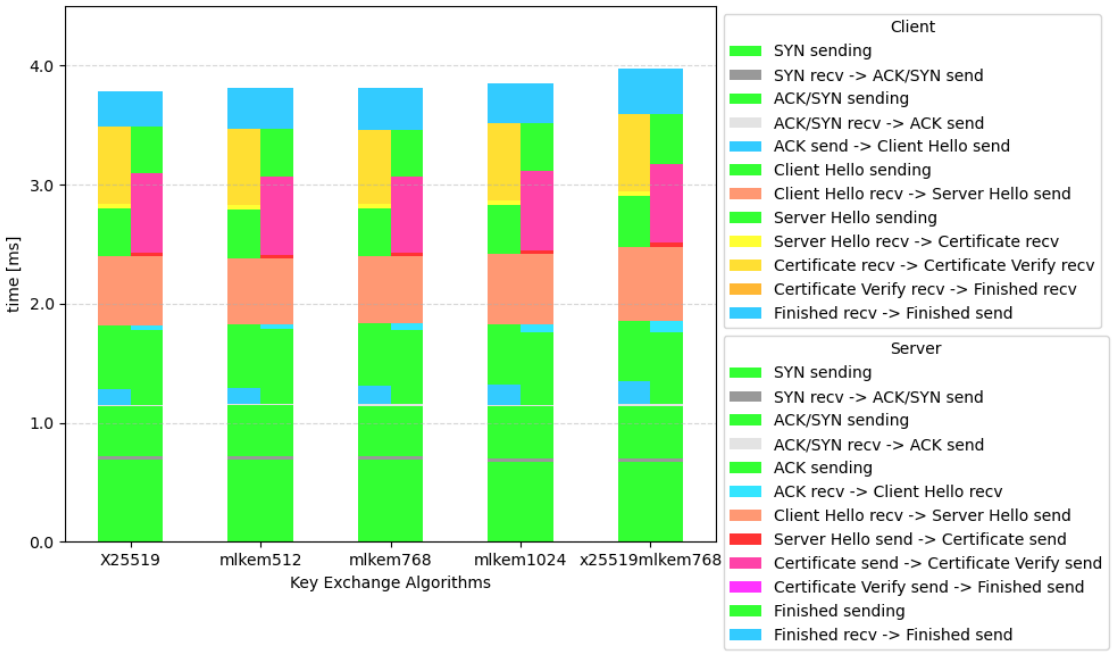

図 16：サーバ及び認証局の署名アルゴリズムが ML-DSA 44 の場合

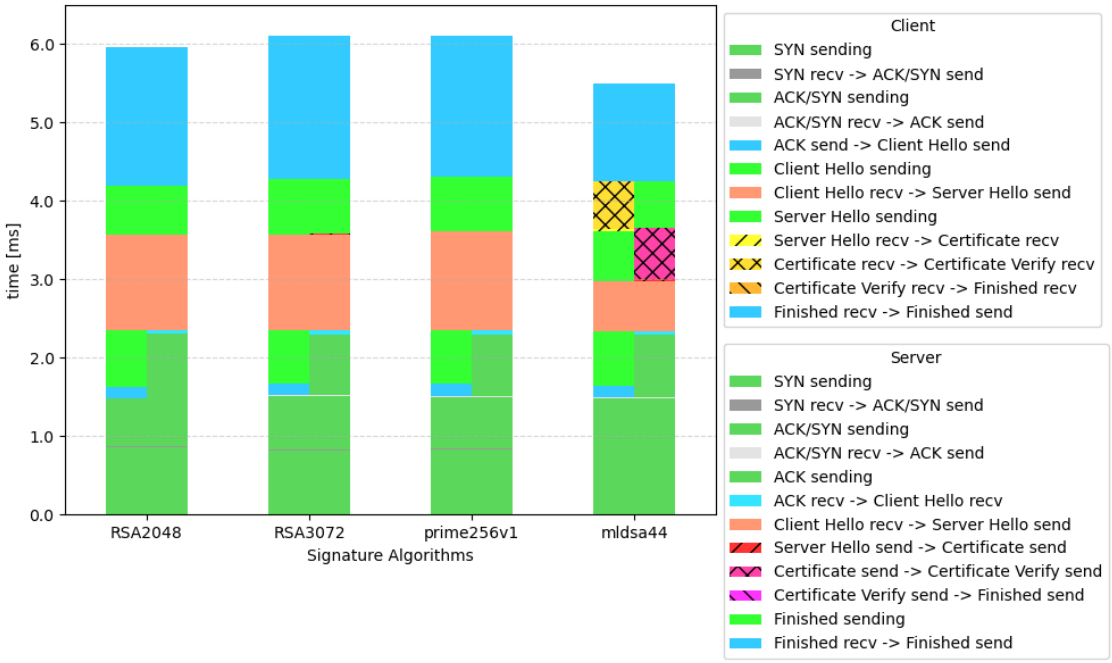

図 17:鍵交換アルゴリズムが X25519、サーバの署名アルゴリズムが SNOVA 24_5_4 の場合

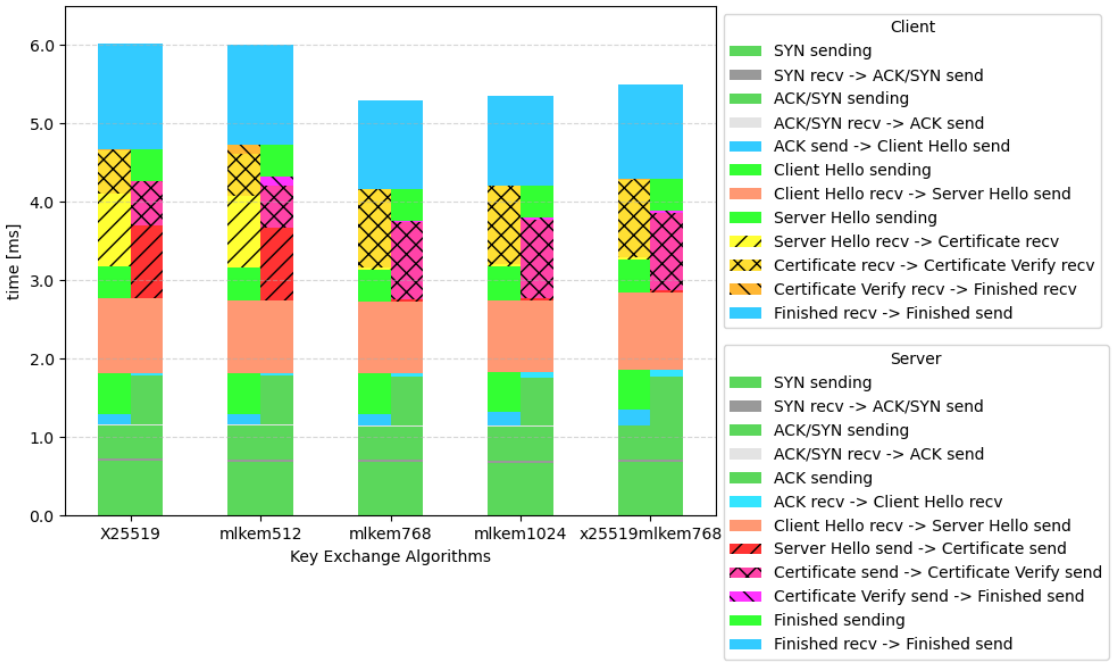

図 18:サーバの署名アルゴリズムが CROSS rsdp 128 balanced、認証局の署名アルゴリズムが ML-DSA 44 の場合

凡例の読み方

ここでは、レポート中に掲載している図の凡例の意味を説明する。

ハンドシェイクの流れ

まず、TCP 及び TLS 1.3 のハンドシェイクは以下の流れで行われる：

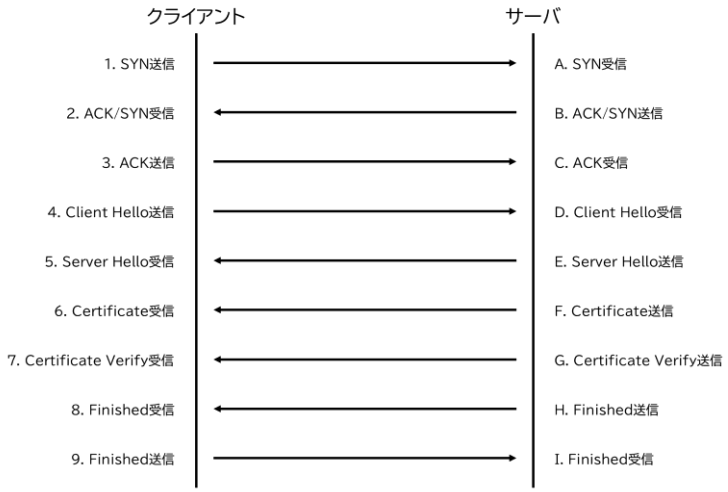

図 19：ハンドシェイクの流れ

このとき、クライアントからみると、以下のような内訳となる。：

凡例上の表記	図 19 中の流れ	意味
SYN sending	1.から A.	SYN パケットの伝送時間
SYN recv -> ACK/SYN send	A.から B.	SYN パケットの受信から ACK/SYN パケット送信までの処理時間
ACK/SYN sending	B.から 2.	ACK/SYN パケットの伝送時間
ACK/SYN send -> ACK send	2.から 3.	ACK/SYN パケットの受信から ACK パケットの送信までの処理時間
ACK send -> Client Hello send	3.から 4.	ACK パケットの送信から Client Hello 送信までの処理時間
Client Hello sending	4.から D.	Client Hello の伝送時間
Client Hello recv -> Server Hello send	D.から E.	Client Hello の受信から Server Hello の送信までの処理時間
Server Hello sending	E.から 5.	Server Hello の伝送時間
Server Hello recv -> Certificate recv	5.から 6.	Server Hello の受信から Certificate の受信までの時間

Certificate recv -> Certificate Verify recv	6.から 7.	Certificate の受信から Certificate Verify の受信までの時間
Certificate Verify recv -> Finished recv	7.から 8.	Certificate Verify の受信から Finished の受信までの時間
Finished recv -> Finished send	8.から 9.	Finished の受信から Finished の送信までの処理時間

クライアントは Finished と同時に HTTP リクエスト等を送信するため実質的にサーバの Finished を受信した時点(8.)でハンドシェイクは確立していると言える。また、この時点以降ではもはや耐量子計算機暗号を用いる場面は無い。そのため、9.から I.は含めないこととする。

同様に、サーバからみると、以下のような内訳となる:

凡例上の表記	図 19 中の流れ	意味
SYN sending	1.から A.	SYN パケットの伝送時間
SYN recv -> ACK/SYN send	A.から B.	SYN パケットの受信から ACK/SYN パケット送信までの処理時間
ACK/SYN sending	B.から 2.	ACK/SYN パケットの伝送時間
ACK/SYN send -> ACK send	2.から 3.	ACK/SYN パケットの受信から ACK パケットの送信までの処理時間
ACK sending	3.から C.	ACK パケットの伝送時間
ACK recv -> Client Hello recv	C.から D.	ACK パケットの受信から Client Hello の受信までの時間
Client Hello recv -> Server Hello send	D.から E.	Client Hello の受信から Server Hello の送信までの処理時間
Server Hello send -> Certificate send	E.から F.	Server Hello の送信から Certificate の送信までの処理時間
Certificate send -> Certificate Verify send	F.から G.	Certificate の送信から Certificate Verify の送信までの処理時間
Certificate Verify send -> Finished send	G.から H.	Certificate Verify の送信から Finished の送信までの処理時間
Finished sending	H.から 8.	Finished の伝送時間
Finished recv -> Finished send	8.から 9.	Finished の受信から Finished の送信までの処理時間

凡例

以下の積み上げグラフの例を用いて説明する：

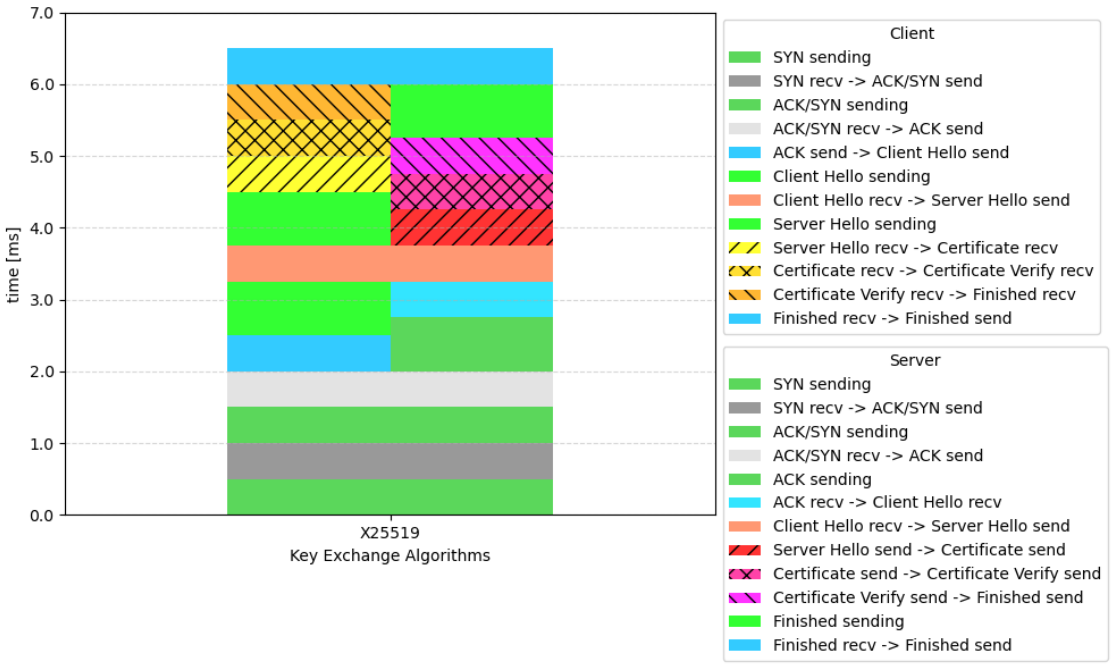

図 20:積み上げグラフの例

積み上げグラフは大きく左右の二つに分かれている。左がクライアントから見たときのハンドシェイクの内訳となっており、右はサーバから見たときのハンドシェイクの内訳となっている。凡例の配色について、基本的には以下の通りである：

色	意味
緑系統	通信に掛かる時間
灰系統	TCP 層に掛かる時間
青系統	クライアント側の処理にかかる時間またはクライアントからのデータを待つ時間
赤系統	サーバ側の処理にかかる時間
黄系統	サーバからのデータを待つ時間

表 13:凡例の配色

赤系統・黄系統の色については見やすさのためにハッチングを入れている。

また、実際の積み上げグラフにおいては特定のパートが潰れていることがある。これは、非常に短い時間であったあるいは TLS のレベルでは同時であったことを意味している。前者は TCP ハンドシェイクの処理、後者は Server Hello から Certificate や Certificate Verify から Finished 等によくみられる。

使用した証明書のサイズ

基礎検証で用いたルート証明書(DER 形式)のサイズは以下の通りである:

2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
998	1254	601	4198

表 14:ルート証明書のサイズ(単位はバイト)

また、サーバ証明書(DER 形式)のサイズは以下の通りである:

	2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
2048-bit RSA	952	1080	759	3112
3072-bit RSA	1080	1208	887	3240
P-256	749	877	556	2909
ML-DSA 44	1992	2120	1800	4152
MAYO 1	2097	2225	1904	4257
MAYO 2	5589	5717	5396	7749
CROSS rsdp 128 balanced	757	885	565	2917
CROSS rsdp 128 fast	757	885	564	2917
CROSS rsdp 128 small	757	885	565	2917
CROSS rsdpg 128 balanced	734	862	542	2894
CROSS rsdpg 128 fast	734	862	541	2894
CROSS rsdpg 128 small	734	862	542	2894
OV Is pkc	67257	67385	67065	69417
OV Ip pkc	44253	44381	44061	46413
OV Is pkc skc	67257	67385	67064	69417
OV Ip pkc skc	44253	44381	44060	46413
SNOVA_24_5_4	1693	1821	1499	3853
SNOVA_24_5_4_SHAKE	1693	1821	1501	3853
SNOVA_24_5_4_esk	1693	1821	1500	3853
SNOVA_24_5_4_SHAKE_esk	1693	1821	1500	3853
SNOVA_37_17_2	10519	10647	10326	12679
SNOVA_25_8_3	2997	3125	2804	5157

表 15:サーバ証明書のサイズ(単位はバイト)

尚、参考までに、実証実験で LB 上に設定された証明書のサイズは、サーバ証明書は 1772 バイト、ルート証明書は 1334 バイトであった。

署名アルゴリズムの性能

今回使用した署名アルゴリズムの鍵及び署名のサイズは以下の通りであった。尚、幅があるものは今回の基礎検証での実測値に依る：

	検証鍵	署名鍵	署名
2048-bit RSA	294	1213 - 1219	256
3072-bit RSA	422	1791 - 1795	384
P-256	91	138	69 - 72
ML-DSA 44	1312	2560	2420
MAYO 1	1420	24	454
MAYO 2	4912	24	186
CROSS rsdp 128 balanced	77	32	13152
CROSS rsdp 128 fast	77	32	18432
CROSS rsdp 128 small	77	32	12432
CROSS rsdpg 128 balanced	54	32	9120
CROSS rsdpg 128 fast	54	32	11980
CROSS rsdpg 128 small	54	32	8960
OV Is pkc	66576	348704	96
OV Ip pkc	43576	237896	128
OV Is pkc skc	66576	32	96
OV Ip pkc skc	43576	32	128
SNOVA_24_5_4	1016	48	248
SNOVA_24_5_4_SHAKE	1016	48	248
SNOVA_24_5_4_esk	1016	36848	248
SNOVA_24_5_4_SHAKE_esk	1016	36848	248
SNOVA_37_17_2	9842	48	124
SNOVA_25_8_3	2320	48	165

表 16：署名アルゴリズムの鍵及び署名のサイズ（単位はバイト）

また、鍵生成及び署名の生成・検証にかかった時間の平均は以下の通りであった：

	鍵生成	署名	検証
2048-bit RSA	40.100951	0.284450	0.037349
3072-bit RSA	141.447273	0.743106	0.057842
P-256	0.035991	0.027631	0.080259
ML-DSA 44	0.021171	0.056068	0.021976
MAYO 1	0.061277	0.175670	0.072550
MAYO 2	0.039809	0.090721	0.024631
CROSS rsdp 128 balanced	0.019268	0.485797	0.345765
CROSS rsdp 128 fast	0.018651	0.294497	0.189645
CROSS rsdp 128 small	0.018674	0.950938	0.698674
CROSS rsdpg 128 balanced	0.013867	0.361315	0.250333
CROSS rsdpg 128 fast	0.013854	0.205371	0.135947
CROSS rsdpg 128 small	0.013743	0.693002	0.483051
OV Is pkc	1.018500	0.026620	0.084897
OV Ip pkc	0.628457	0.026382	0.068714
OV Is pkc skc	1.035813	0.588346	0.085262
OV Ip pkc skc	0.637256	0.421898	0.069517
SNOVA_24_5_4	0.079901	0.376327	0.073716
SNOVA_24_5_4_SHAKE	0.091771	0.383104	0.084514
SNOVA_24_5_4_esk	0.081209	0.312946	0.069761
SNOVA_24_5_4_SHAKE_esk	0.094355	0.313012	0.084244
SNOVA_37_17_2	0.226378	0.294003	0.048625
SNOVA_25_8_3	0.083274	0.256646	0.058010

表 17：鍵生成及び署名の生成・検証にかかった時間（単位はミリ秒）