Supplementary Material for Proof of Concept Conducted at Daiwa Securities Group

Disclaimer

This document summarizes current information on Post-Quantum Cryptography (PQC) and the results of verifications conducted within our group. The content is subject to change depending on future standardization trends, product implementation status, and changes in the threat environment.

The information provided in this document is for informational purposes only. Product names, company names, and service names mentioned herein are included for the purpose of explaining the demonstration environment, and do not constitute a guarantee or recommendation by our company for the adoption, performance, or future compatibility of any specific product or service.

The technical trends and product compatibility status described in this document are generally based on information as of March 2026.

Unless otherwise noted, all figures and tables in this document were created by Daiwa Institute of Research Ltd.

Supplementary Information on Basic Verification Using a Verification Environment

1. Verification Environment

The terminal used for verification was an Amazon EC2 instance. Its details are as follows:

Item	Content
Instance Type	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
Memory	4GiB

In addition, the following libraries were used to measure the performance of individual algorithms:

Library Name	Overview
Cryptography (Python)	Version 46.0.3. Used to measure the performance of RSA, X25519, and P-256.
liboqs	Version 0.14.0. Used to measure the performance of each post-quantum cryptography algorithm. Also, `liboqs-python` was used as a wrapper.

Furthermore, the following software programs were used for specific communications:

Name	Overview
OpenSSL	Version 3.5.4. Used as a client and server for TLS communication.
OQS Provider	Version 0.10.1-dev. Used for TLS communication using some post-quantum cryptography algorithms.

Note that `pwntools` and `paramiko` were also used for client-side implementation, but they were only used for communications unrelated to the handshake, and TLS communication was performed using OpenSSL.

2. Notes

The versions of each software are the latest versions at the time the experiment started (as of November 2025).

At the time of writing, the latest versions are 46.0.5 for cryptography, 0.15.0 for liboqs, 3.6.1/3.5.5 for OpenSSL, and 0.11.0 for OQS Provider.

Also, OpenSSL 3.6.0 was available at the time the experiment started, but 3.5.4, which is an LTS (Long Term Support) version, was selected.

3. Verification Items and Experiment Overview

First, to measure the performance of pure key exchange algorithms, the time required for key generation, encapsulation, and decapsulation (in the case of X25519, the generation of the public key and shared secret) was investigated.

The algorithms verified were X25519, ML-KEM 512, ML-KEM 768, and ML-KEM 1024. Note that although HQC was selected as a standardization target, it was excluded this time because its FIPS draft has not been published.

For each algorithm, the time required for key generation, encapsulation, and decapsulation was measured 1,000 times, and the results were calculated based on the data from 999 times, excluding the first execution which is affected by overhead.

Next, to measure the performance of pure signature algorithms, the time required for key generation, signature generation, and signature verification was investigated.

The algorithms verified were 2048-bit RSA, 3072-bit RSA, P-256, and ML-DSA 44. Note that Falcon was excluded this time because the final version of FIPS had not been approved as of January 2026 despite being selected as a standardization target, and SLH-DSA was excluded because it is not suitable for practical use in terms of performance.

For each algorithm, the time required for key generation, signature generation, and signature verification was measured 1,000 times, and the results were calculated based on the data from 999 times, excluding the first execution which is affected by overhead.

Then, in actual communication, the time required to establish a TLS handshake was investigated. This communication was performed between EC2 instances in two different availability zones, with one acting as a client and the other as a server. At this time, the server was required to present the server certificate and the certificate authority (CA) certificate prepared for the basic verification.

The key exchange algorithms verified were X25519, ML-KEM 512, ML-KEM 768, ML-KEM 1024, and X25519MLKEM768. In addition, 2048-bit RSA, 3072-bit RSA, P-256, and ML-DSA 44 were used as signature algorithms for the server and the certificate authority. Note that the “server’s signature algorithm" here refers to the algorithm used by the server for signing in the Certificate Verify message. The “certificate authority’s signature algorithm" refers to the signature algorithm attached to the certificate.

The number of handshakes was set to 1,000, and the analysis was performed based on the communication data.

In the results presented later, communication time is calculated by the difference between the times recorded by the client and the server. Note that the EC2 instances running the client and server were synchronized with the same NTP server (169.254.169.123), and the time recorded in the capture files was corrected based on the deviation from the NTP server.

Also, for reference, some results are included for cases where MAYO, CROSS, UOV, and SNOVA—which are currently being considered as standardization candidates—were adopted as the server's signature algorithm.

4. Results

Figures appearing in the main text are compiled together in the section entitled “List of Figures.”

4-1. Key Exchange Algorithms

4-1-1. Performance Measurement Results

First, the sizes of the public key, private key, ciphertext, and secret are as follows:

	Public Key	Private Key	Ciphertext	Secret
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

Table 1: Sizes of public key, private key, ciphertext, and secret (in bytes)

Next, regarding the performance of pure algorithms, the mean, standard deviation, quartiles, and robust coefficient of variation[1] of the time taken for key generation were as follows. For time distribution, please refer to Figure 1:

[1] “Robust coefficient of variation" is calculated by dividing the interquartile range by the median. This is due to the asymmetry of the distribution.

	Mean	Standard Deviation	1st Quartile	Median	3rd Quartile	Robust CV
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

Table 2: Time statistics taken for key generation of key exchange algorithms (Mean, standard deviation, and quartiles are in milliseconds)

Next, the mean, standard deviation, quartiles, and robust coefficient of variations in time taken for encapsulation were as follows. For time distribution, please refer to Figure 2:

	Mean	Standard Deviation	1st Quartile	Median	3rd Quartile	Robust CV
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

Table 3: Statistics of time taken for encapsulation (Mean, standard deviation, and quartiles are in milliseconds)

Finally, the mean, standard deviation, quartiles, and robust coefficient of variations in time taken for decapsulation were as follows. For time distribution, please refer to Figure 3:

	Mean	Standard Deviation	1st Quartile	Median	3rd Quartile	Robust CV
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

Table 4: Statistics of time taken for decapsulation (Mean, standard deviation, and quartiles are in milliseconds)

4-1-2. Evaluation of Measurement Results

The average time taken for key generation, encapsulation, and decapsulation is on the order of tens of microseconds. In contrast, the time taken to establish a TLS handshake is on the order of milliseconds, so the difference in time taken for cryptographic processing can be considered within the margin of error for normal communication purposes.

If we were to dare to evaluate it, the time taken for key generation and encapsulation of ML-KEM can be said to be faster and more stable than X25519 based on the results of the mean and robust coefficient of variation. On the other hand, regarding decapsulation, it is considered that ML-KEM 512 tends to have an upward deviation based on the shape of the distribution in Figure 3 and the results of the robust coefficient of variation.

In addition, from the perspective of computational resources such as CPU usage and memory usage, the load of all algorithms is extremely low, and we believe that the load of cryptographic processing will not be a problem if the specifications are equal to or higher than the terminal used this time.

The sizes of the public key, private key, and ciphertext of ML-KEM are more than 20 times those of X25519.

In TLS handshake applications, the public key is sent in the Client Hello and the ciphertext is sent in the Server Hello to the communication partner, so if the public key and ciphertext are large, the number of packets required for transmission will increase accordingly.

Since the de facto standard for the maximum segment size of TCP is 1460 bytes (or 1440 bytes), ML-KEM 1024 exceeds this, and the number of TCP packets required for Client Hello and Server Hello increases. However, two TCP packets were sufficient to send the Client Hello and Server Hello.

4-2. Signature Algorithms

4-2-1. Performance Measurement Results

First, the sizes of the verification key, signature key, and signature are as follows. For those with a range, it depends on the actual measured values in this basic verification:

	Public Key	Private Key	Signature
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

Table 5: Sizes of verification key, signature key, and signature (in bytes)

Next, regarding the performance of pure algorithms, the mean, standard deviation, quartiles, and robust coefficient of variation of the time taken for key generation were as follows. For time

distribution, please refer to Figure 4 and Figure 5:

	Mean	Standard Deviation	1st Quartile	Median	3rd Quartile	Robust CV
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

Table 6: Statistics of time taken for key generation of signature algorithms (Mean, standard deviation, and quartiles are in milliseconds)

Next, the mean, standard deviation, quartiles, and robust coefficient of variation of the time taken for signature generation were as follows. For time distribution, please refer to Figure 6:

	Mean	Standard Deviation	1st Quartile	Median	3rd Quartile	Robust CV
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

Table 7: Statistics of time taken for signature generation (Mean, standard deviation, and quartiles are in milliseconds)

Finally, the mean, standard deviation, quartiles, and robust coefficient of variation of the time taken for verification were as follows. For time distribution, please refer to Figure 7:

	Mean	Standard Deviation	1st Quartile	Median	3rd Quartile	Robust CV
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

Table 8: Statistics of time taken for signature verification (Mean, standard deviation, and quartiles are in milliseconds)

4-2-2. Evaluation of Measurement Results

Regarding the time taken for key generation, it can be seen that both the mean and the robust coefficient of variation for RSA are considerably larger compared to the others. In other words, it takes time to generate, and the time has a relatively wide range. However, at present, keys are not generated for each TLS handshake, so it is not considered a significant problem from the perspective of TLS communication.

The time taken for signature generation and verification affects the time taken to generate the Certificate Verify on the server side and to verify the Certificate/Certificate Verify on the client side.

The time taken for signature generation with RSA is more than 10 times that of P-256, and more than 5 times that of ML-DSA 44. However, since it is less than 1 millisecond, it cannot be said to be fatally slow for the client unless there is a demand to establish communication extremely quickly. On the other hand, for the server, changing the signature algorithm can save computational resources, albeit slightly. If handshakes are performed frequently, there is room for consideration.

Also, from the shape of the distribution in Figure 6 and the results of the robust coefficient of variation, it is considered that the time taken for signature generation of ML-DSA 44 tends to have a slight upward deviation.

The time taken for signature verification is the slowest for P-256, but it is on the order of tens of microseconds, and the robust coefficient of variation is not that large, so it can be said that there is no significant difference regardless of which algorithm is selected for normal use.

If we were to dare to evaluate it, ML-DSA 44 can be said to be faster and more stable than conventional cryptography based on the results of the mean and robust coefficient of variation.

Since the public key is included in the server certificate and the signature is sent in the Certificate Verify, if these sizes are large, the number of TCP packets sent by the server will also increase accordingly.

Regarding the number of TCP packets used by the server to send from Server Hello to Finished in the basic verification, an excerpt of the results is as follows:

CA	2048-bit RSA			ML-DSA 44	
Server	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

Table 9: Number of TCP packets used to send from Server Hello to Finished (Excerpt)

It should be noted that this value depends not only on the signature algorithm but also on the contents of the certificate and the presence or absence of an intermediate certificate. For the size of the certificates used in the basic verification, please refer to “Size of Certificates Used.”

If the communication quality is not ideal, TCP retransmission may occur. If the number of TCP packets increases, the possibility of TCP retransmission occurring during the handshake increases. It will be necessary to keep this in mind if the communication after the handshake is established is lightweight and you want to keep latency as low as possible. In addition, it is also necessary to pay attention to the limitations of the congestion window.

4-3. TLS Handshake

4-3-1. Measurement Results

For specific verification results, please refer to the separately distributed CSV file.

4-3-2. Evaluation and Hypotheses on Measurement Results

There are several facts to note when evaluating the time required to establish a handshake. A comparison of the conventional signature algorithms (3072-bit RSA and P-256) and the post-quantum cryptography ML-DSA 44 will be discussed at the end of this report.

For simplicity, the average time obtained from measurement results will hereafter simply be referred to as “time.”

4-3-2-1. When the Key Exchange Algorithm is X25519 and the CA Signature Algorithm is 2048-bit RSA

Figure 8 shows the results of the time taken for each part when the server's signature algorithm is varied, given that the key exchange algorithm is X25519 and the CA signature algorithm is 2048-bit RSA.

First, regarding UOV and SNOVA, a delay at the time of the TCP handshake is noticeable. It is unlikely that the choice of cryptography used in the subsequent TLS handshake affects the TCP handshake, and since MAYO and CROSS show no such effect, it cannot be considered an effect of using a oqs-provider. The cases where UOV and SNOVA were adopted as the server's signature algorithm were conducted on different days using other algorithms, which might have affected the communication quality.

In cases where RSA, P-256, ML-DSA 44, MAYO, and CROSS were adopted as the server's signature algorithm, the time taken to send the SYN packet was 0.69 ms, the ACK/SYN packet was 0.43 ms, and the ACK packet was 0.63 ms during the TCP handshake. Therefore, it can be said that the communication itself takes approximately 1 ms round-trip.

As an overall trend, compared to conventional RSA and P-256, the time required to establish a handshake becomes longer when post-quantum cryptography is used.

As individual trends, the following characteristics are observed depending on the algorithm:

Point 1. There is almost no time between sending the Server Hello and sending the Finished message (RSA, P-256, ML-DSA 44, etc.).

Point 2. The time between sending the Server Hello and sending the Certificate is noticeable (MAYO 2, SNOVA 25_8_3, UOV).

Point 3. The time between sending the Certificate and sending the Certificate Verify is noticeable (CROSS rsdp 128, SNOVA 37_17_2).

Point 4. The time between sending the Certificate Verify and sending the Finished message is noticeable (CROSS rsdpg 128 fast).

Point 5. The time between receiving the Client Hello and sending the Server Hello is short (MAYO 2, UOV, SNOVA 37_17_2/25_8_3).

Point 6. The time between receiving the Finished message and sending the Finished message is short (RSA, CROSS rsdp 128 balanced/fast, SNOVA 37_17_2/25_8_3, etc.).

We will discuss these points, but first, let's confirm the behavior of OpenSSL.

Regarding OpenSSL's behavior[2] when writing transmission data to the send buffer: first, if the size of the transmission data is less than or equal to the available space in the send buffer, it stores the data in the buffer and finishes.

If not, and the send buffer is not empty, it first fills the send buffer. After filling the send buffer, it flushes the data in the send buffer.

If the send buffer is empty (including after a flush) and the transmission data is larger than the send buffer size, it continues to write directly to the underlying BIO without going through the send buffer. If transmission data smaller than the send buffer size remains after writing to the underlying BIO, or if the transmission data was originally smaller than the send buffer size, it stores it in the send buffer and finishes.

Note that since the write size to the underlying BIO is not limited to the send buffer size, even if the size of the transmission data is not a multiple of the send buffer size, the remainder does not necessarily stay in the send buffer. Also, the default send buffer size is 4096 bytes.

Based on this specification, let's first consider Point 1. This point means that the messages from Server Hello to Finished are sent almost simultaneously. In particular, it is considered that the transmission of the Server Hello was delayed until the Finished message was ready to be sent.

From OpenSSL's specifications, messages are not sent until the size of the data the server can send

[2] openssl/crypto/bio/bf_buff.c

reaches 4096 bytes or the generation of the Finished message is completed. Therefore, in this case, it is considered that the total size of each message sent by the server was 4096 bytes or less.

This point will be specifically confirmed in “4-3-2-1-2. The Relationship between Transmission Timing and Size of Each Message.”

Regarding Point 2, since the Server Hello is sent before the Finished message is generated, it is considered that the send buffer has been filled. In this basic verification, the Server Hello was at most 1663 bytes, so the send buffer cannot be completely filled by the Server Hello alone. Therefore, it is considered that the flushed data in the send buffer included the Server Hello and at least a part of the Certificate.

And from the fact that the transmission timing of the Server Hello and the Certificate was not simultaneous, it is considered that only a part of the Certificate was sent, and the rest was not sent. There are two possible reasons for this:

2-1 The rest of the Certificate was stored in the send buffer.

2-2 The server intended to send the rest of the Certificate as well, but could not do so for another reason.

In the case of 2-1, from OpenSSL's specifications, it is considered that the size of the remaining Certificate was 1 byte or more and less than 4096 bytes, meaning the total size of the Server Hello and the Certificate was greater than 4096 bytes and less than 8192 bytes.

This point will be specifically confirmed in “4-3-2-1-2. The Relationship between Transmission Timing and Size of Each Message,” but for MAYO 2 and SNOVA 25_8_3 in Figure 8, the total sizes of the Server Hello and Certificate are 6788 bytes and 4196 bytes, respectively.

On the other hand, UOV is considered to fall under 2-2. This point will be discussed in “4-3-2-1-1. The Relationship between Transmission Timing of Each Message and Congestion Window.”

The case of Point 3 is basically the same as Point 2; it is considered that the Certificate was sent because the send buffer was filled. At this time, there are both cases where the Server Hello and the Certificate were sent simultaneously and cases where they were not.

First, consider the case where the Server Hello and the Certificate were sent simultaneously. At this time, since it does not fall under Point 2, it is considered that the total size of the Server Hello and the Certificate was 4096 bytes or less, or 8192 bytes or more.

Furthermore, the Certificate Verify must not have been sent. Similar to Point 2, there are two possible reasons for this:

3-1 The Certificate Verify was stored in the send buffer.

3-2 The server intended to send the Certificate Verify as well, but was hindered.

Regarding 3-2, it will be discussed in “4-3-2-1-1. The Relationship between Transmission Timing of Each Message and Congestion Window,” but CROSS rsdp 128 is considered to fall under 3-2.

Next, in the case of 3-1 where the total size of the Server Hello and the Certificate is 4096 bytes or less, a part of the Certificate Verify remains in the send buffer and waits for the generation of the Finished message. In this case, the time from the transmission of the Certificate to the transmission of the Certificate Verify was within the range of 20 to 30 microseconds. It is inferred that the generation of the Finished message itself is fast, and in this case, it is considered that it did not result in a very noticeable difference.

Finally, in the case of 3-1 where the total size of the Server Hello and the Certificate is 8192 bytes or more, the Certificate Verify was less than 4096 bytes. In fact, for SNOVA 37_17_2 in Figure 8, the total size of the Server Hello and the Certificate is 11718 bytes, and the Certificate Verify is 154 bytes.

Also, in the case of Point 3, if the Server Hello and the Certificate were not sent simultaneously, i.e., if it falls under Point 2, it was limited to the cases falling under 3-2 above. This will be discussed in "4-3-2-1-1. The Relationship between Transmission Timing of Each Message and Congestion Window."

In Point 4, it is considered that the entire Certificate Verify was sent without waiting for the Finished message. On the other hand, since Finished is the last message, it is sent as soon as it is generated. And as discussed in the consideration of Point 3, it is inferred that the generation of the Finished message takes about tens of microseconds. Therefore, for Point 4 to be clearly observed, the transmission of the Finished message must be hindered for some reason after the transmission of the Certificate Verify.

Possible reasons are the congestion window limit and the Nagle algorithm, but since OpenSSL enables TCP_NODELAY by default[3], the congestion window is considered to be the cause. Details will be discussed in "4-3-2-1-1. The Relationship between Transmission Timing of Each Message and Congestion Window."

Point 5 is considered a consequence of Point 1, Point 2, and Point 3.

First, if the total size of the Server Hello and the Certificate is 4096 bytes or more, the generation of the Certificate itself does not take much time[4], so it is considered that it was practically sent as soon as the generation of the Server Hello was completed. In other words, this transmission timing is considered the original transmission timing of the Server Hello.

On the other hand, if the total size of the Server Hello and the Certificate is less than 4096 bytes, it is considered that the transmission timing was delayed because it waited for the generation of subsequent messages.

---

[3] openssl/apps/lib/s_sockets.c

[4] This is because the Certificate simply sends the server certificate and certificate authority certificate that have already been registered.

From these, it is considered that a formal recorded difference appeared in the time from receiving the Client Hello to sending the Server Hello depending on the algorithm.

Regarding Point 6, it can basically be explained by the difference in the transmission timing of each message due to Point 2, Point 3, and Point 4.

First, when the client receives the Finished message from the server, it verifies the message authentication code and then generates its own Finished message in the same way. However, depending on the reception timing of each message, the client might still be processing messages prior to the Certificate Verify even when it receives the Finished message.

If the transmission timing of each message shifts due to Point 2, Point 3, and Point 4, a difference may occur between the client's reception of the Finished message and the reception of messages prior to the Certificate Verify. At this time, since the client has already started processing the messages prior to the Certificate Verify when it receives the Finished message, it is considered that the recorded time was shortened.

However, for RSA, exceptional circumstances are suggested. This point will be confirmed in "4-3-2-2-1. When the Server and the Certificate Authority Use the Same Signature Algorithm."

4-3-2-1-1. The Relationship between Transmission Timing of Each Message and Congestion Window

First, let's confirm the specifications of the congestion window, which is one of the congestion controls in TCP communication.

RFC 5681[5] defines the congestion window as the amount of data a sender can transmit unilaterally in TCP communication, and states that the congestion window should be increased every time an ACK packet for new data is received. RFC 6928[6] sets the initial value of the congestion window to 10 MSS and states that it should not be increased by ACK packets or SYN/ACK packets during the TCP handshake. Also, Section 4.4 of RFC 7661[7] states that if the sender is not using even half of the congestion window, the congestion window should not be updated.

In the TLS handshake, the first new data sent by the server is the Server Hello, so at the time of sending the Server Hello, the server can send up to a maximum of 10 MSS. Note that depending on the OS implementation, it might be managed by the number of segments rather than the number of bytes.[8] In this case, even if the size of the transmission data has not reached 10 MSS, transmission

[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
[8] linux/net/ipv4/tcp_cubic.c

will be restricted if 10 packets are sent.

Note that this basic verification is IPv4 communication, and since the timestamp option is enabled, 1 MSS is 1448 bytes.

Based on this specification, let's consider behavior shown in 2-2 in the previous section.

First, based on OpenSSL's specifications, if the total size of the Server Hello and the Certificate is 8192 bytes or more, it is considered that the rest of the Certificate will be sent after the Server Hello and a part of the Certificate are sent.

However, if 11 or more packets are required to send the Server Hello and the Certificate, the transmission will be restricted by the congestion window during the transmission of the Certificate, and it will have to wait for an ACK packet from the client.

If the total size of the Server Hello and the Certificate is 8192 bytes or more, the first flushed 4096 bytes will use 3 packets, and the rest of the Certificate will be divided and sent every 1448 bytes.

Therefore, if the total size of the Server Hello and the Certificate exceeds 14232 bytes, it is considered that it will be restricted by the congestion window during the transmission of the Certificate, causing a time difference between sending the Server Hello and sending the Certificate.

This point will be confirmed in “4-3-2-1-2. The Relationship between Transmission Timing and Size of Each Message.”

Similarly, it is also possible to be restricted by the congestion window during the transmission of the Certificate Verify (3-2 in the previous section). Considering OpenSSL's specifications, the situations where the Certificate Verify is being sent are as follows:

Situation 1 The total size of the Server Hello and the Certificate is less than 4096 bytes.

Situation 2 The total size of the Server Hello and the Certificate is 4096 bytes or more and less than 8192 bytes.

Situation 3 The total size of the Server Hello and the Certificate is 8192 bytes or more and 12784 bytes or less.

The 12784 bytes in Situation 3 means that the Server Hello and the Certificate fit within 9 packets.

To be restricted by the congestion window in the above situations, the following corresponding conditions are required respectively:

Condition 1. The total size from the Server Hello to the Certificate Verify exceeds 14232 bytes.

Condition 2. The total size from the Server Hello to the Certificate Verify exceeds 13984 bytes.

Condition 3. The size of the Certificate Verify exceeds the value obtained by multiplying the number of transmittable packets by 1448 bytes.

Note that all combinations satisfying these conditions adopted CROSS as the signature algorithm.

When Condition 1 is met, the time taken from the transmission of the Certificate to the transmission

of the Certificate Verify was 1.03 ms or more. When Condition 3 is met, it was 0.98 ms or more. Therefore, in these cases, it is considered that the transmission of the Certificate Verify was hindered due to the restriction of the congestion window.

On the other hand, even when Condition 2 is met, most of the time taken from the transmission of the Certificate to the transmission of the Certificate Verify is 0.60 ms or less, and some combinations are even under 0.10 ms.

When Condition 2 is met, the transmission of the Certificate waits for the generation of the Certificate Verify. Therefore, a time difference occurs between the transmission of the Server Hello and the transmission of the Certificate Verify, and during that time, an ACK packet from the client for the initially sent 4096 bytes may be received. It is considered that this causes the congestion window to slide, making it less likely to be restricted by the congestion window between the transmission of the Certificate and the transmission of the Certificate Verify.

Note that looking at the time taken from the transmission of the Server Hello to the transmission of the Certificate Verify, it was 1.04 ms or more when Condition 1 was met, 0.99 ms or more when Condition 2 was met, and 1.01 ms or more when Condition 3 was met.

Finally, let's consider the case where the transmission of the Finished message is restricted by the congestion window. Two situations can be considered for this:

1. The Certificate Verify was sent without waiting for the generation of the Finished message, and the congestion window limit was reached exactly at the Certificate Verify.
2. The Certificate Verify was sent simultaneously with the Finished message, but the Finished message was divided, and the congestion window limit was reached exactly at the first half of the Finished message.

Since the reception of ACK packets causes the congestion window to slide or increase, it cannot be judged solely by the size of each message. Therefore, we will individually check the cases where the transmission of the Finished message is considered to have been restricted by the congestion window.

First, if the number of packets required to send from the Server Hello to the Finished message is 10 or less, it is within the initial value of the congestion window, so it is not restricted by the congestion window. In this case, the time taken from the transmission of the Certificate Verify to the transmission of the Finished message was within 0.04 ms. Therefore, for combinations that took 0.04 ms or more, it is suggested that the transmission of the Finished message might have been restricted by the congestion window.

Next, in the combinations where the Finished message was divided during this basic verification, the number of packets required to send from the Server Hello to the Finished message was 7, so there were no cases where the second half of the Finished message could not be sent due to the congestion window restriction.

The combinations of algorithms where the time taken from the transmission of the Certificate Verify to the transmission of the Finished message was 0.04 ms or more can be divided into those taking 0.12 ms, 0.57 ms, and 1.01 ms or more. The number of packets required to send from the Server Hello to the Certificate Verify was 16 packets, 13 packets, and 10 packets, respectively.

Conversely, even if the number of packets required to send from the Server Hello to the Certificate Verify was 10/13/16 packets, the Finished message was not always restricted by the congestion window. Below, we consider this difference.

First is the case where the number of packets required to send from the Server Hello to the Certificate Verify is 10 packets. In the combinations of algorithms where the Finished message was not restricted by the congestion window in this case, the total of the Server Hello and the Certificate was 4096 bytes or more. On the other hand, when the Finished message was restricted by the congestion window, it was less than 4096 bytes.

If the total of the Server Hello and the Certificate is 4096 bytes or more, a part of the Server Hello and the Certificate is flushed, so there is a possibility of receiving an ACK packet from the client before sending the Finished message. Therefore, it is considered that there is room in the congestion window at the time of sending the Finished message. On the other hand, if it is less than 4096 bytes, the messages from the Server Hello to the Certificate Verify are sent almost simultaneously, so there is no opportunity to receive an ACK packet, and since 10 packets are used from the Server Hello to the Certificate Verify, it is considered that the Finished message is restricted by the congestion window.

Next is the case where the number of packets required to send from the Server Hello to the Certificate Verify is 13 packets. Similarly in this case, in the combinations of algorithms where the total of the Server Hello and the Certificate was 4096 bytes or more, the Finished message was not restricted by the congestion window.

However, when it was less than 4096 bytes, there were cases where it was restricted by the congestion window and cases where it was not. This was when the key exchange algorithm was ML-KEM512 and the CA signature algorithm was ML-DSA 44; when the server's signature algorithm was CROSS rsdpg 128 balanced, it was not restricted, and when it was CROSS rsdpg 128 small, it was restricted.

The difference between these two is the difference in the time required to generate the signature, and CROSS rsdpg 128 small is 0.33 ms slower than CROSS rsdpg 128 balanced. That is, in the case of CROSS rsdpg 128 small, the timing when the Certificate Verify becomes transmittable is relatively delayed. As a result, the possibility of receiving an ACK packet from the client for the initially sent 4096 bytes before the Certificate Verify becomes transmittable increases.

When an ACK packet from the client is received, since the number of packets already sent is less than half of the congestion window, the congestion window is not updated. As a result, only 10 packets

can be sent, and the limit is reached with the rest of the Certificate and the Certificate Verify. Therefore, it is considered that the Finished message was restricted by the congestion window.

On the other hand, for CROSS rsdpg 128 balanced, the timing when it becomes transmittable is relatively early, so the Certificate Verify may become transmittable before receiving an ACK packet from the client. Therefore, it is easy to be restricted by the congestion window during the transmission of the Certificate Verify. It is considered that when the Finished message becomes transmittable, the congestion window has slid or increased, making it less likely to be restricted.

Finally, there is the case where the number of packets required to send from the Server Hello to the Certificate Verify is 16 packets. In this case, conversely, in the combinations of algorithms where the total of the Server Hello and the Certificate was less than 4096 bytes, the Finished message was not restricted by the congestion window. In this combination, since it is restricted by the congestion window during the transmission of the Certificate Verify, it is considered that when the Finished message becomes transmittable, the congestion window has slid or increased, making it less likely to be restricted.

On the other hand, in the combinations where the total of the Server Hello and the Certificate is 4096 bytes or more, there is a possibility of receiving an ACK packet from the client during the transmission of the Certificate Verify. If the number of transmitted packets is more than half of the congestion window at the time of receiving the ACK packet, the congestion window increases to 13. Since this 13 is reached by the rest of the Certificate and the Certificate Verify, it is considered that the Finished message was restricted by the congestion window.

4-3-2-1-2. The Relationship between Transmission Timing and Size of Each Message

First, Figure 9 and Figure 10 show the relationship between the total size of each message sent by the server and the time taken from the transmission of the Server Hello to the transmission of the Finished message.

When the total size of each message was 4096 bytes or less, the time taken from the transmission of the Server Hello to the transmission of the Finished message was 0.005 ms or less, whereas when it exceeded 4096 bytes, it exceeded 0.02 ms.

From this difference, in the case of algorithm combinations where the total size of each message is 4096 bytes or less, it is considered that the transmission of the Server Hello is delayed until the Finished message becomes transmittable.

Next, Figure 11 and Figure 12 show the relationship between the total size of the Server Hello and the Certificate and the time taken from the transmission of the Server Hello to the transmission of the Certificate.

When the size of the Server Hello and the Certificate is less than 4096 bytes, the time taken from the transmission of the Server Hello to the transmission of the Certificate is 0.005 ms or less. From this, it is considered that the timing when these two messages become transmittable is almost simultaneous.

When the size of the Server Hello and the Certificate is 4096 bytes or more and less than 8192 bytes, the time taken from the transmission of the Server Hello to the transmission of the Certificate is 0.44 ms or more except for some combinations, so it is considered that the transmission of the Certificate was delayed until the Certificate Verify became transmittable.

Note that “some combinations" refers to the case where P-256 was adopted as the server's signature algorithm. At this time, the time taken from the transmission of the Server Hello to the transmission of the Certificate was 0.10 ms. From this, it is also suggested that the generation of the Certificate Verify is fast when P-256 is adopted as the server's signature algorithm.

Also, in Figure 11, there are algorithms taking 0.90 ms to 1.50 ms. These correspond to when the server's signature algorithm is 3072-bit RSA, CROSS rsdp 128 balanced/small, or CROSS rsdpg 128 small. Since the signature generation of these algorithms is relatively slow, it is considered that this is reflected here.

When the size of the Server Hello and the Certificate is 8192 bytes or more and 14232 bytes or less, the time taken from the transmission of the Server Hello to the transmission of the Certificate was 0.032 ms or less. Although it takes relatively more time, in these cases, the time taken from the transmission of the Certificate to the transmission of the Certificate Verify is 0.41 ms or more except for some combinations, so it is considered that the Certificate was sent before the Certificate Verify became transmittable.

Note that “some combinations" refers to the case where P-256 was adopted as the server's signature algorithm. At this time, the time taken from the transmission of the Certificate to the transmission of the Certificate Verify was 0.07 ms to 0.08 ms, so even in this case, it is suggested that the generation of the Certificate Verify is fast when P-256 is adopted as the server's signature algorithm.

Finally, when the size of the Server Hello and the Certificate exceeds 14232 bytes, the time taken from the transmission of the Server Hello to the transmission of the Certificate was at least 1.44 ms. Since this time difference is more than one round-trip of TCP communication, it is suggested that it was waiting for an ACK packet from the client.

4-3-2-2. When the Key Exchange Algorithm is X25519 and the CA Signature Algorithm is P-256 or ML-DSA 44

First, Figure 13 shows the results of the time taken for each part when the server's signature algorithm

is varied, given that the key exchange algorithm is X25519 and the CA signature algorithm is P-256. Next, Figure 14 shows the results of the time taken for each part when the server's signature algorithm is varied, given that the key exchange algorithm is X25519 and the CA signature algorithm is ML-DSA 44.

Comparing the case where the CA signature algorithm is 2048-bit RSA (Figure 8) with Figure 13 and Figure 14, the following characteristics are confirmed regarding the time required to establish the handshake:

1. For RSA, P-256, and ML-DSA 44, when the same signature algorithm is used for the server and the CA, it is faster compared to when different signature algorithms are used.
2. For MAYO 1 and SNOVA (excluding 37_17_2), it is faster when the CA signature algorithm is ML-DSA 44.

Below, we consider point 1. Point 2 will be discussed in "4-3-2-4. Acceleration due to the Timing of Certificate Reception."

4-3-2-2-1. When the Server and the Certificate Authority Use the Same Signature Algorithm

For RSA, P-256, and ML-DSA 44 in Figure 8, Figure 13, and Figure 14, differences are seen in the time taken from the reception of the Finished message to the transmission of the Finished message. Since the time taken from the reception of the Finished message to the transmission of the Finished message may include the processing time of previously received messages, we will compare the client's processing time using the time from the reception of the Server Hello to the transmission of the Finished message.

First, when the key exchange algorithm is X25519, it was as follows:

		CA			
		2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
Server	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

Table 10: Time from Reception of Server Hello to Transmission of Finished when Key Exchange Algorithm is X25519 (Unit: ms)

Next, for ML-KEM 512, it was as follows:

		CA			
		2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
Server	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

Table 11: Time from Reception of Server Hello to Transmission of Finished when Key Exchange Algorithm is ML-KEM 512 (Unit: ms)

Finally, for X25519MLKEM768, it was as follows:

		CA			
		2048-bit RSA	3072-bit RSA	P-256	ML-DSA 44
Server	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

Table 12: Time from Reception of Server Hello to Transmission of Finished when Key Exchange Algorithm is X25519MLKEM768 (Unit: ms)

The colored cells in Table 10, Table 11, and Table 12 are combinations where the total size of the Server Hello and the Certificate is 4096 bytes or more and less than 8192 bytes. In these combinations, the Server Hello and a part of the Certificate is sent first, and the rest of the Certificate waits for messages after the Certificate Verify.

In this case, it should be noted that the client experiences a waiting time for the Certificate after receiving and processing the Server Hello, so it tends to be larger than the difference in algorithms compared to other items.

Also, from Table 10, Table 11, and Table 12, when the same signature algorithm is used for the server and the CA, there is a tendency for the client's cryptographic processing time to be shorter compared to other combinations. Below, we consider the reason for this.

When the server and CA use the same signature algorithm, the signature on the server certificate, the signature on the CA certificate, and the signature transmitted in Certificate Verify are all generated by the same algorithm. In this case, when the client verifies the signature in Certificate Verify, it can reuse the instructions already loaded during the verification of the certificate in Certificate. As a result, the overhead for signature verification is reduced, and the time to generate and send Finished is shortened.

Therefore, when measuring the performance impact of changing the signature algorithm of a certificate, it is necessary to consider the influence of the signature algorithms used in intermediate CA certificates and in the various fields within the certificate itself.

4-3-2-3. Changes Due to the Choice of Key Exchange Algorithm

Figure 15 shows the time spent on each part of the handshake when the server's and CA's signature algorithm is 2048-bit RSA and the key exchange algorithm is varied.

The total handshake time shows no extreme difference: 3.69 ms for X25519 and 3.72 ms for ML-KEM 512. On the other hand, the hybrid X25519MLKEM768 takes 3.96 ms, a difference of 0.27 ms compared to X25519.

This difference is primarily attributable to the following: 0.06 ms from sending ACK to sending Client Hello, 0.06 ms from receiving Client Hello to sending Server Hello, and 0.13 ms from receiving Finished to sending Finished.

Note that this difference is not particularly significant for typical use cases.

Figure 16 shows the time spent on each part of the handshake when the server's and CA's signature algorithm is ML-DSA 44 and the key exchange algorithm is varied.

Even when both the key exchange and signature algorithms are lattice-based, no speedup attributable to this combination was observed.

In both Figure 15 and Figure 16, X25519MLKEM768 takes the longest time. This is a natural result: a longer key length means more data to transmit, increasing processing time, and a more complex hybrid construction increases cryptographic processing overhead.

However, it has been confirmed that in certain combinations of signature and key exchange algorithms, a larger key or signature size can paradoxically lead to a faster handshake. This phenomenon is analyzed in the next section.

4-3-2-4. Acceleration due to the Timing of Certificate Reception

Figure 17 shows the results of the time taken for each part when the key exchange algorithm is X25519, the server's signature algorithm is SNOVA 24_5_4, and the certificate authority's (CA) signature algorithm is varied.

In Figure 17, the handshake is established fastest in the case of ML-DSA 44, where the size of the certificate transmitted in the Certificate message is the largest. The total time taken for the handshake is 5.51 milliseconds for ML-DSA 44, and 5.97 milliseconds for the next fastest, 2048-bit RSA. Changing to ML-DSA 44 results in a reduction of 0.46 milliseconds.

The difference between ML-DSA 44 and the others is the timing of the transmission of the Server Hello (and Certificate). The time from when the client sends the Client Hello until it receives the Server Hello is 1.98 milliseconds for ML-DSA 44, and 2.57 milliseconds for the next fastest, 2048-bit RSA.

In contrast, there is not much difference in the time taken for cryptographic processing. In fact, from the server's perspective, the time from receiving the Client Hello to sending the Finished message is 1.22 milliseconds when the CA's signature algorithm is 2048-bit RSA, 1.23 milliseconds for 3072-bit RSA, 1.26 milliseconds for P-256, and 1.32 milliseconds for ML-DSA 44.

Similarly, from the client's perspective, the time from receiving the Server Hello to sending the Finished message is 1.77 milliseconds when the CA's signature algorithm is 2048-bit RSA, 1.81 milliseconds for 3072-bit RSA, 1.79 milliseconds for P-256, and 1.88 milliseconds for ML-DSA 44.

Furthermore, Figure 18 shows the results of the time taken for each part when the server's signature algorithm is CROSS rsdp 128 balanced, the CA's signature algorithm is ML-DSA 44, and the key exchange algorithm is varied.

In Figure 18, the handshake is established faster when the key size is larger. The total time taken for the handshake is 5.29 milliseconds for ML-KEM 768, 5.50 milliseconds for X25519MLKEM768, and 6.02 milliseconds for X25519.

Unlike Figure 17, there is not much difference in the time from the client sending the Client Hello to receiving the Server Hello. The shortest is 1.83 milliseconds for ML-KEM 768, and the longest is 1.91 milliseconds for X25519MLKEM768.

On the other hand, including the reception of the Certificate, it is 1.91 milliseconds for X25519MLKEM768, whereas it is 2.82 milliseconds for X25519.

Also, the time from when the client receives the Certificate to when it sends the Finished message is 1.91 milliseconds for X25519 and ML-KEM 512, 2.13 milliseconds for ML-KEM 768, 2.14 milliseconds for ML-KEM 1024, and 2.21 milliseconds for X25519MLKEM768. Considering that the processing time of the Server Hello might not be included for X25519 and ML-KEM 512 due to the timing of the Server Hello reception, it cannot be said that there is a significant difference in the cryptographic processing time.

It is considered that the earlier timing of the Certificate reception allows the client to start cryptographic processing early while concurrently waiting for messages from Certificate Verify onwards, thereby accelerating the overall handshake. This is consistent even when the key exchange algorithm is X25519 and the server's signature algorithm is SNOVA 24_5_4.

4-3-2-5. Comparison of 3072-bit RSA/P-256/ML-DSA 44

Since the security of 2048-bit RSA is 112-bit security, there is a need to migrate to a more secure option by 2030, regardless of quantum resistance. As migration targets, 3072-bit RSA, P-256, and ML-DSA 44 (or other various ECDSAs with equivalent or higher security), which have 128-bit security, are considered.

The performance of the pure algorithms is as already described, and it cannot be said that there is a clear difference regardless of which one is selected, at least when there are computational resources equivalent to or greater than those in this basic verification.

From the perspective of the time taken to establish a handshake, as discussed in the previous sections, it can be accelerated depending on the combination of algorithms. Therefore, a sweeping statement cannot be made.

A provisional evaluation is given from the following two perspectives:

1. Compare the cases where one signature algorithm is used as the CA's signature algorithm and the remaining two are adopted as the server's signature algorithm.
2. Compare the cases where the same signature algorithm is adopted for both the server and the CA.

In both of these perspectives, the average time taken to establish the handshake was shortest for P-256, followed by ML-DSA 44, and longest for 3072-bit RSA. Therefore, in an ideal communication environment, it can be said that ML-DSA 44 can establish a handshake faster than 3072-bit RSA.

However, it should be noted that the number of packets transmitted by the server increases as the size of the ciphertext and signature increases. Since there is a possibility of packet loss when routing through the Internet, the 1-millisecond difference between ML-DSA 44 and 3072-bit RSA could be reversed depending on the packet loss rate. In addition, the larger size of the server certificate and intermediate CA certificate may lead to hitting the congestion window limit.

List of Figures

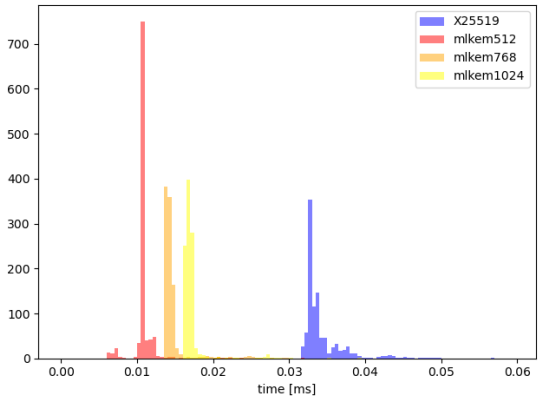

Figure 1: Distribution of time taken for key generation of key exchange algorithms

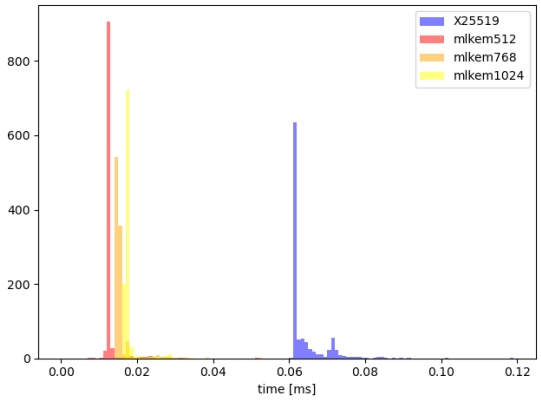

Figure 2: Distribution of time taken for encapsulation

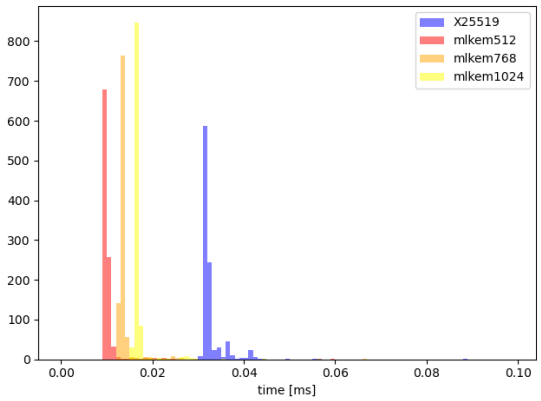

Figure 3: Distribution of time taken for decapsulation

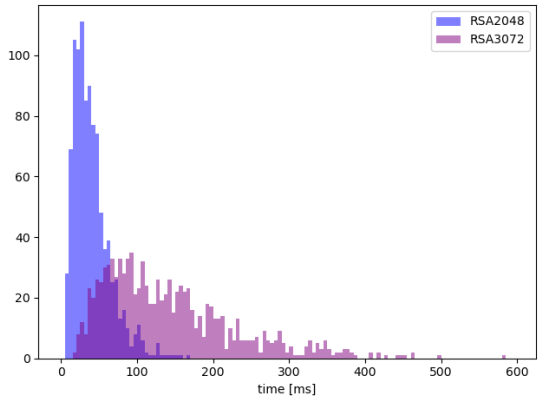

Figure 4: Distribution of time taken for key generation of signature algorithms (RSA)

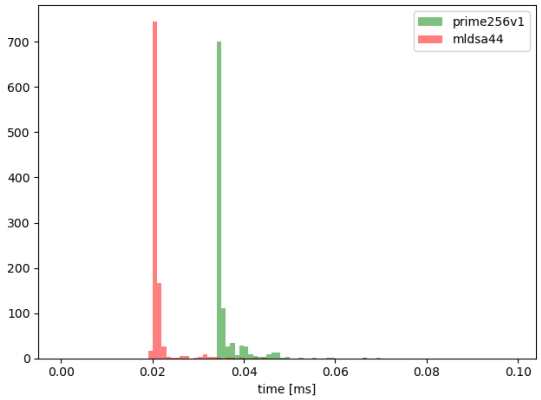

Figure 5: Distribution of time taken for key generation of signature algorithms (P-256, ML-DSA 44)

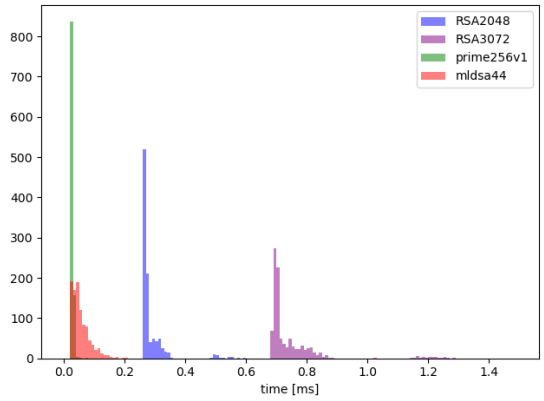

Figure 6: Distribution of time taken for signature generation

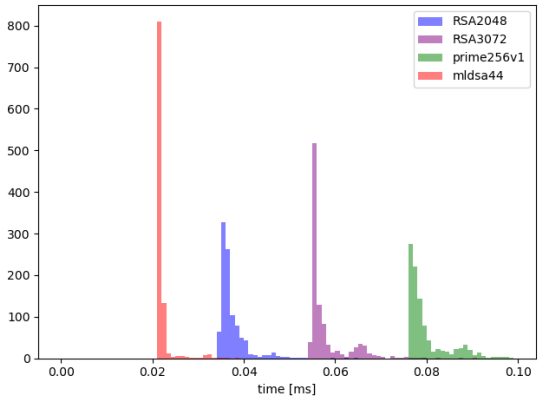

Figure 7: Distribution of time taken for signature verification

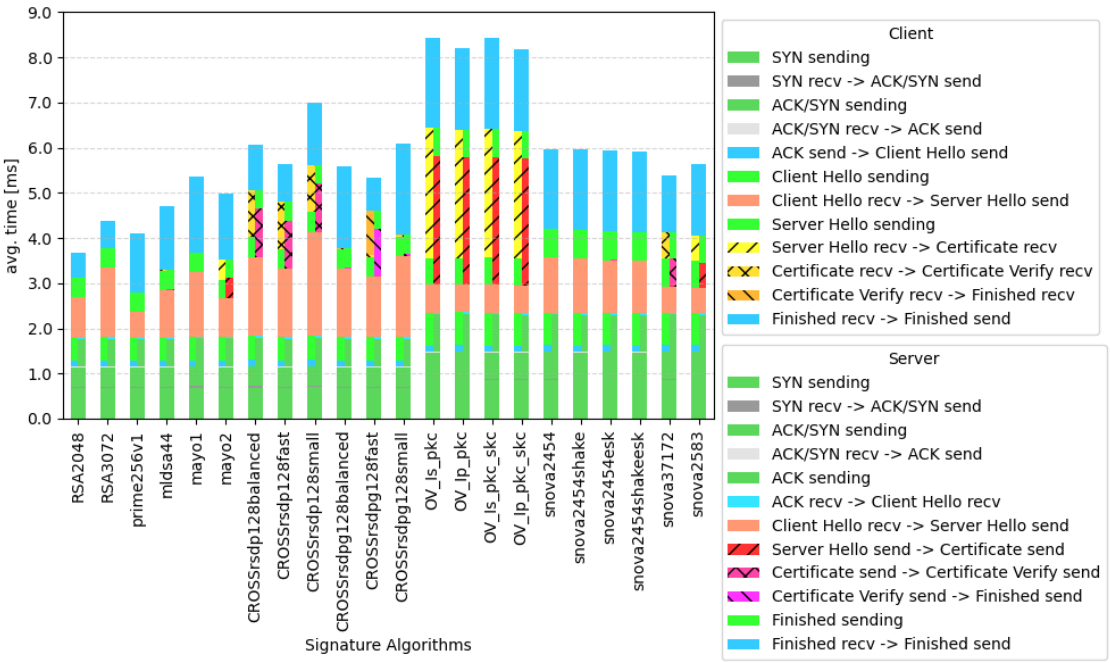

Figure 8: Case where the key exchange algorithm is X25519 and the CA's signature algorithm is 2048-bit RSA

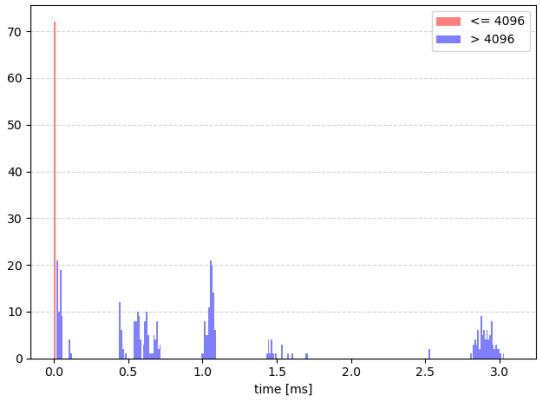

Figure 9: Relationship between the total size of each message sent by the server and the time taken from sending Server Hello to sending Finished

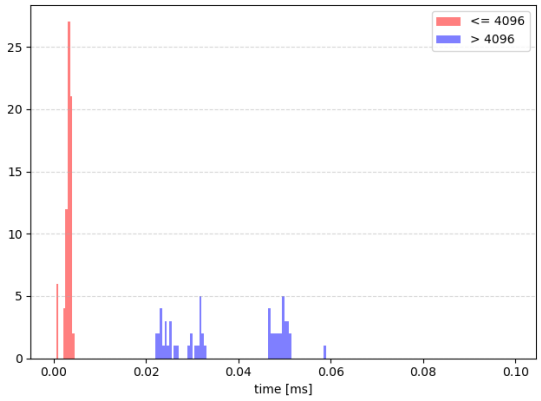

Figure 10: Figure 9 restricted to the range of 0.0 ms to 0.1 ms

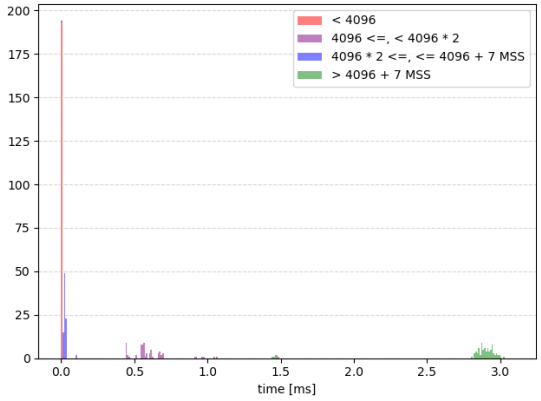

Figure 11: Relationship between the total size of Server Hello and Certificate and the time taken from sending Server Hello to sending Certificate

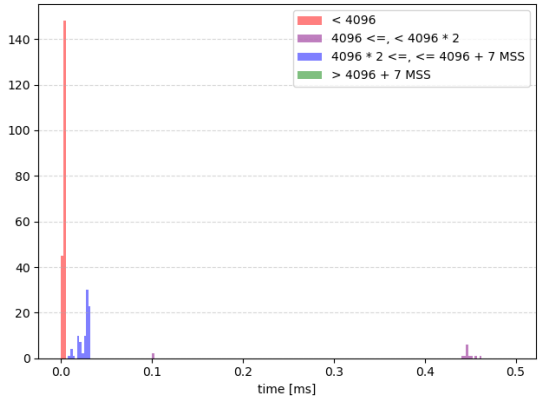

Figure 12: Figure 11restricted to the range of 0.0 ms to 0.5 ms

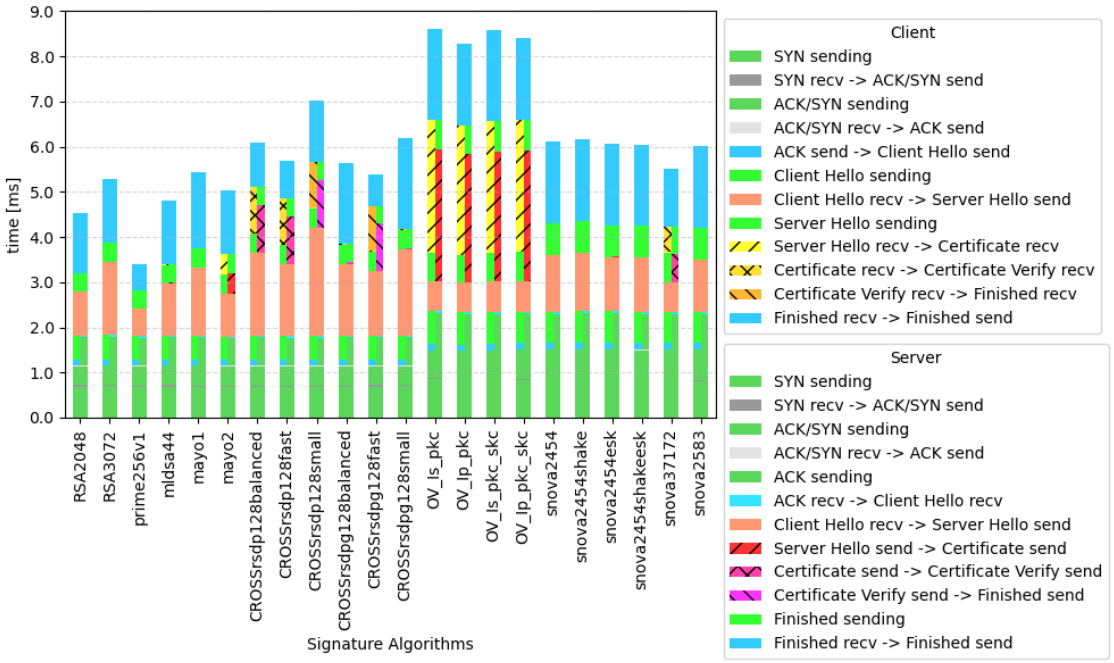

Figure 13: Case where the key exchange algorithm is X25519 and the CA's signature algorithm is P-256

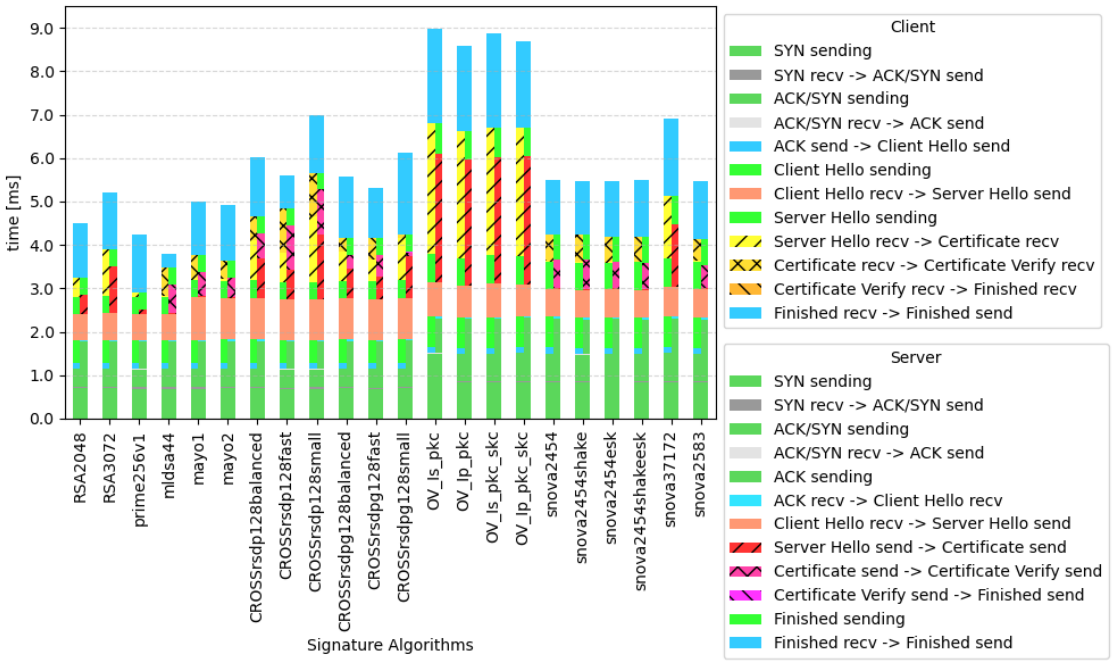

Figure 14: Case where the key exchange algorithm is X25519 and the CA's signature algorithm is ML-DSA 44

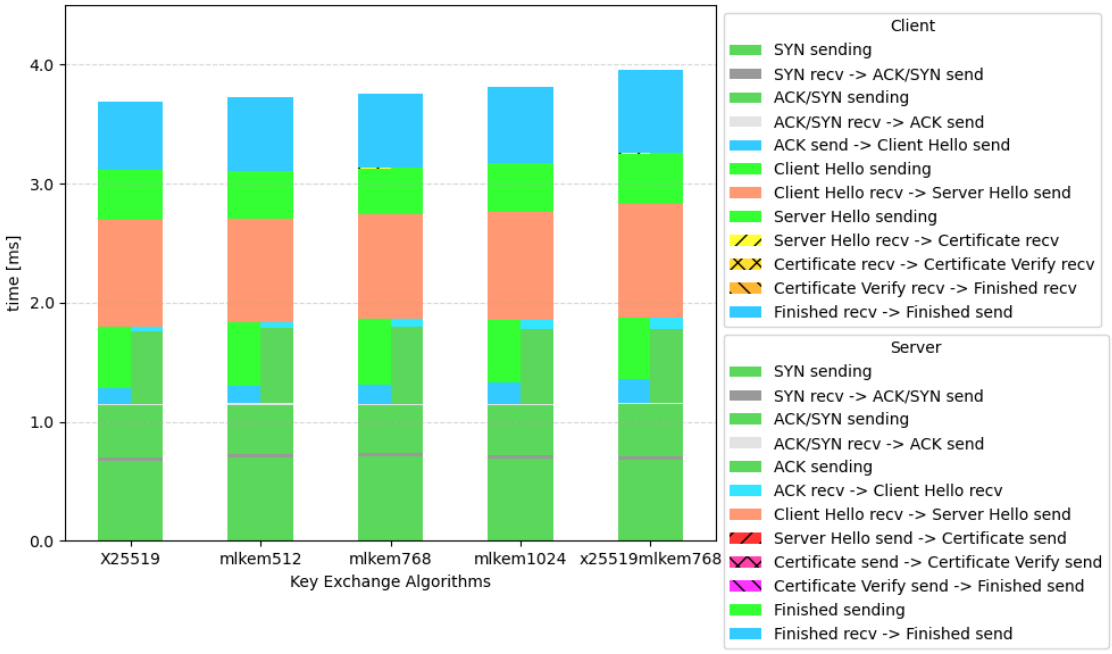

Figure 15: Case where the server's and CA's signature algorithms are 2048-bit RSA

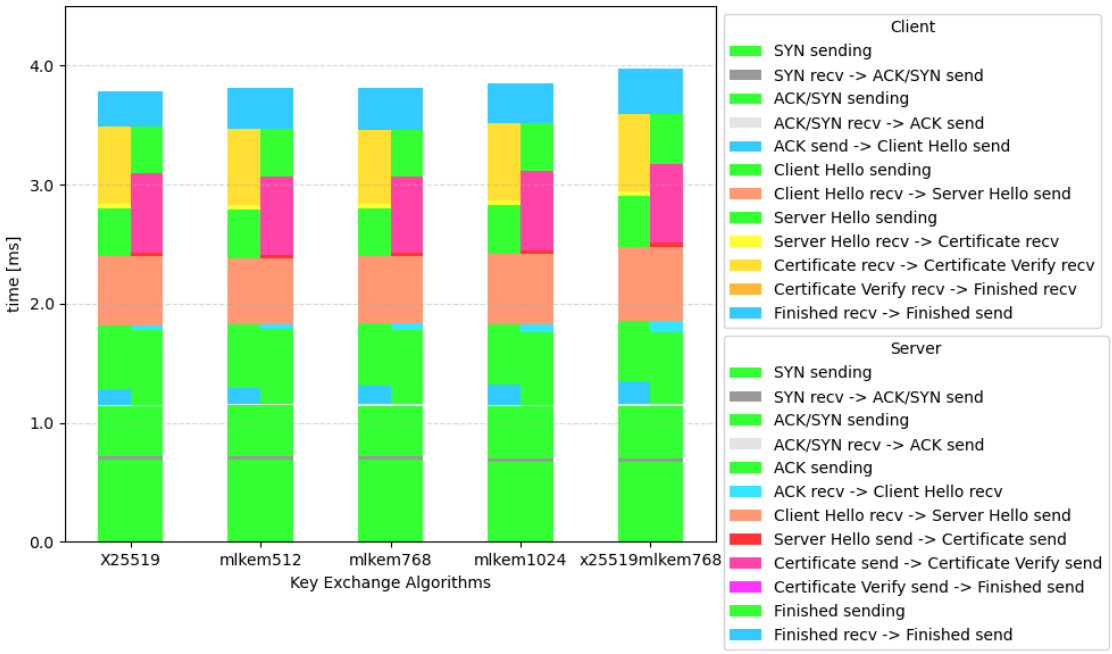

Figure 16: Case where the server's and CA's signature algorithms are ML-DSA 44

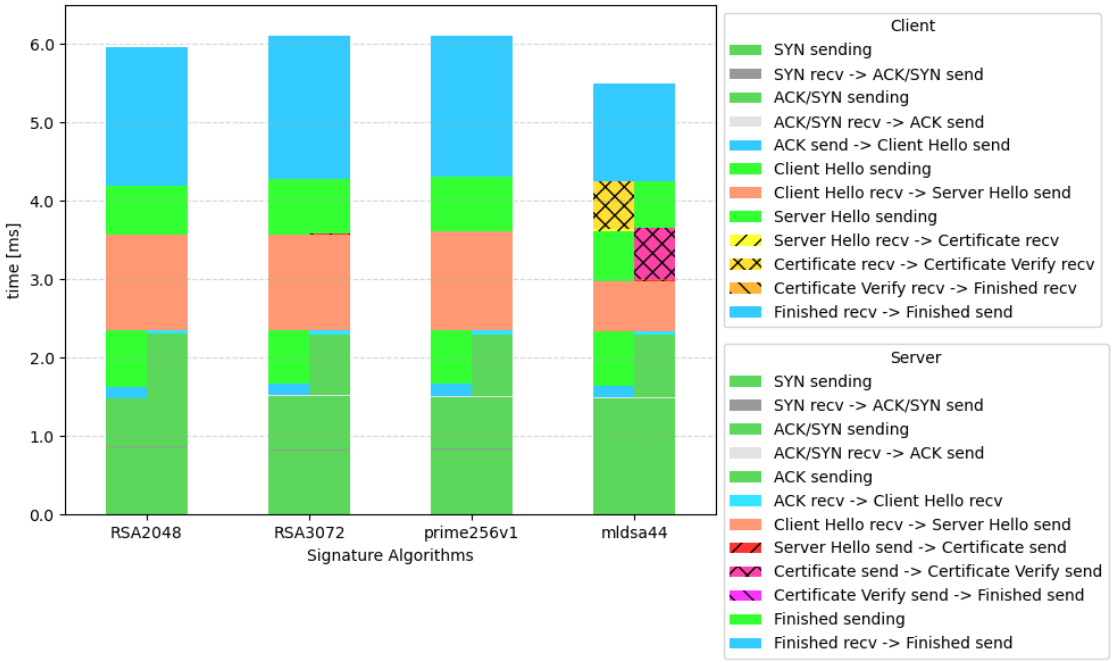

Figure 17: Case where the key exchange algorithm is X25519 and the server's signature algorithm is SNOVA 24_5_4

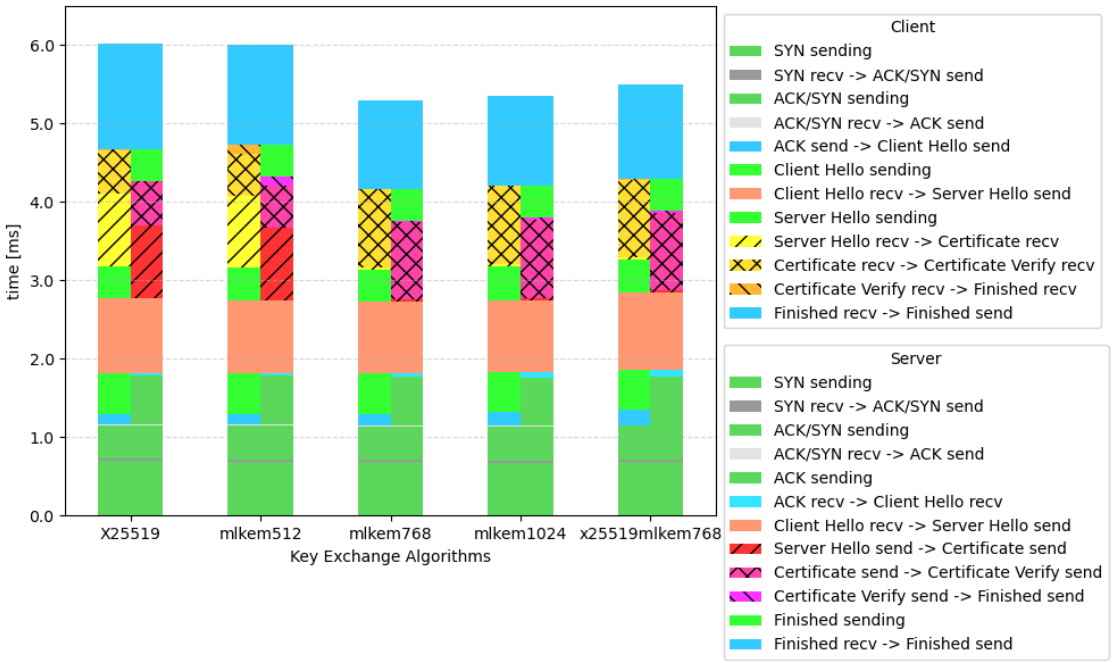

Figure 18: Case where the server's signature algorithm is CROSS rsdp 128 balanced and the CA's signature algorithm is ML-DSA 44

How to Read the Legends

Here, we explain the meaning of the legends in the figures included in this report.

Handshake Flow

First, the TCP and TLS 1.3 handshakes are performed in the following flow:

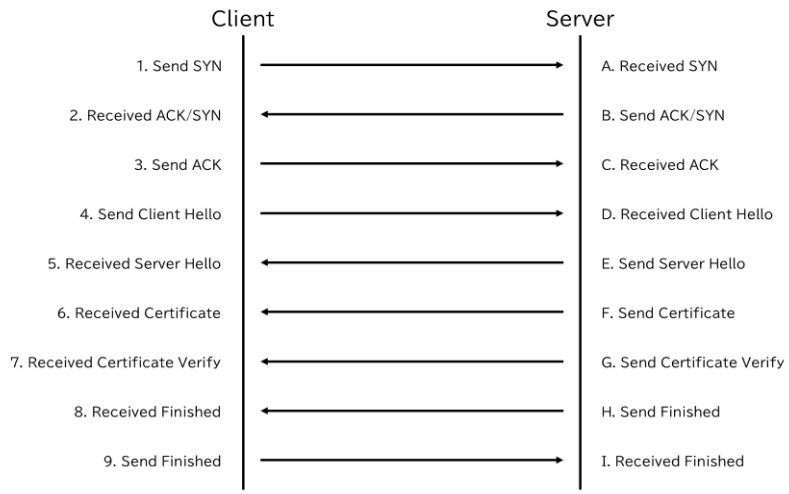

Figure 19: Handshake Flow

At this time, from the client's perspective, the breakdown is as follows:

Legend Notation	Flow in Figure 19	Meaning
SYN sending	1. to A.	Transmission time of the SYN packet
SYN recv -> ACK/SYN send	A. to B.	Processing time from receiving the SYN packet to sending the ACK/SYN packet
ACK/SYN sending	B. to 2.	Transmission time of the ACK/SYN packet
ACK/SYN send -> ACK send	2. to 3.	Processing time from receiving the ACK/SYN packet to sending the ACK packet
ACK send -> Client Hello send	3. to 4.	Processing time from sending the ACK packet to sending the Client Hello
Client Hello sending	4. to D.	Transmission time of the Client Hello
Client Hello recv -> Server Hello send	D. to E.	Processing time from receiving the Client Hello to sending the Server Hello
Server Hello sending	E. to 5.	Transmission time of the Server Hello
Server Hello recv -> Certificate recv	5. to 6.	Time from receiving the Server Hello to receiving the Certificate
Certificate recv -> Certificate Verify recv	6. to 7.	Time from receiving the Certificate to receiving the

		Certificate Verify
Certificate Verify recv -> Finished recv	7. to 8.	Time from receiving the Certificate Verify to receiving the Finished
Finished recv -> Finished send	8. to 9.	Processing time from receiving the Finished to sending the Finished

Since the client sends HTTP requests simultaneously with the Finished message, the handshake is practically established when the server's Finished message is received. Also, there is no longer any use for post-quantum cryptography after this point. Therefore, subsequent steps are not included.

Similarly, from the server's perspective, the breakdown is as follows:

Legend Notation	Flow in Figure 19	Meaning
SYN sending	1. to A.	Transmission time of the SYN packet
SYN recv -> ACK/SYN send	A. to B.	Processing time from receiving the SYN packet to sending the ACK/SYN packet
ACK/SYN sending	B. to 2.	Transmission time of the ACK/SYN packet
ACK/SYN send -> ACK send	2. to 3.	Processing time from receiving the ACK/SYN packet to sending the ACK packet
ACK sending	3. to C.	Transmission time of the ACK packet
ACK recv -> Client Hello recv	C. to D.	Time from receiving the ACK packet to receiving the Client Hello
Client Hello recv -> Server Hello send	D. to E.	Processing time from receiving the Client Hello to sending the Server Hello
Server Hello send -> Certificate send	E. to F.	Processing time from sending the Server Hello to sending the Certificate
Certificate send -> Certificate Verify send	F. to G.	Processing time from sending the Certificate to sending the Certificate Verify
Certificate Verify send -> Finished send	G. to H.	Processing time from sending the Certificate Verify to sending the Finished
Finished sending	H. to 8.	Transmission time of the Finished
Finished recv -> Finished send	8. to 9.	Processing time from receiving the Finished to sending the Finished

Legend

This is explained using the following stacked graph example:

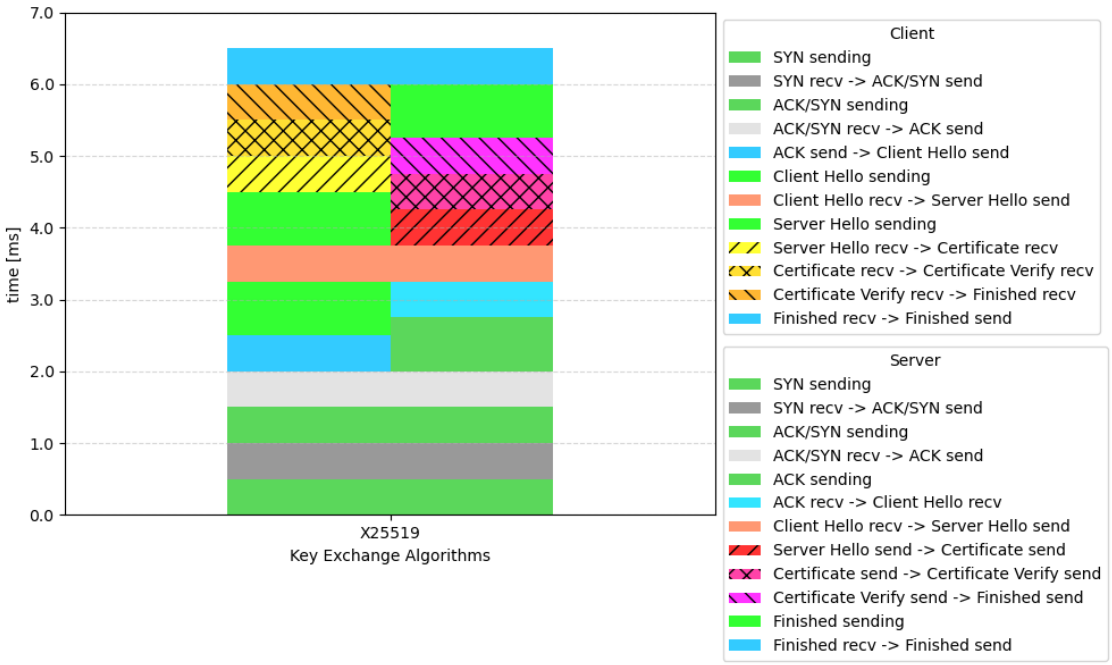

Figure 20: Example of a stacked graph

The stacked graph is broadly divided into left and right sections. The left shows the breakdown of the handshake from the client's perspective, and the right shows it from the server's perspective. The color scheme of the legend is basically as follows:

Color	Meaning
Green shades	Time taken for communication
Gray shades	Time taken in the TCP layer
Blue shades	Processing time on the client side or time waiting for data from the client
Red shades	Processing time on the server side
Yellow shades	Time waiting for data from the server

Table 13: Legend Color Scheme

Hatching is added to the red and yellow shades for better visibility.

Also, in the actual stacked graphs, certain parts may appear collapsed. This means that the time was extremely short or simultaneous at the TLS level. The former is often seen in TCP handshake processing, and the latter in transitions like Server Hello to Certificate or Certificate Verify to Finished.

Size of Certificates Used

The sizes of the root certificates (DER format) used in the basic verification are as follows:

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

Table 14: Root Certificate Sizes (in bytes)

The sizes of the server certificates (DER format) are as follows:

	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

Table 15: Server Certificate Sizes (in bytes)

For reference, the sizes of the certificates configured on the LB in the demonstration experiment were 1772 bytes for the server certificate and 1334 bytes for the root certificate.

Performance of Signature Algorithms

The sizes of the keys and signatures for the signature algorithms used were as follows. Note that ranges depend on actual measurements in this basic verification:

	Public Key	Private Key	Signature
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

Table 16: Key and Signature Sizes for Signature Algorithms (in bytes)

Also, the average times taken for key generation, signature generation, and verification were as follows:

	Key Generation	Signature Generation	Signature Verification
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

Table 17: Time Taken for Key Generation and Signature Generation/Verification (in milliseconds)