CELS (Crystalline Encryption Layered Security): a Security Extension of Messaging Applications Using Post-Quantum Cryptography

Noah Loke, Vaibhav Bajpai and Tingting Chen

March 17, 2025

CELS (Crystalline Encryption Layered Security): A Security Extension of Messaging Applications using Post-Quantum Cryptography

Noah Loke
Computer Science Department
Cal Poly Pomona
Pomona, CA, USA
contact@noahloke.com

Vaibhav Kumar Bajpai
Independent Researcher
Seattle, WA, USA
vaibhav.iiith@gmail.com

Tingting Chen
Computer Science Department
Cal Poly Pomona
Pomona, CA, USA
tingtingchen@cpp.edu

***Abstract*—Privacy and security are vital for digital communications to protect sensitive information and enable free expression. In this paper we present CELS (Crystalline Encryption Layered Security), a security extension based on a hybrid cryptographic approach that layers post-quantum cryptography (PQC) on top of a messaging application, to provide end-to-end data security. We introduce two CELS protocols, one for shared key establishment and one for key retrieval. Our prototype implementation, Discord CELS is a Firefox sidebar extension that adds end-to-end message encryption to a Discord server channel with native webhooks. It uses CRYSTALS-Kyber-1024 compiled with Emscripten for key encapsulation along with AES-256 GCM and PBKDF2 from the Web Crypto API for data encryption. CELS is resistant to attacks by both classical and quantum computers. CELS enables users to protect their messages with end-to-end encryption from being viewed by the messaging application company when using their existing service such as Discord. It is adaptable to other messaging applications and has cross-device flexibility, subject to platform constraints. Extensive performance evaluations demonstrate that the prototype system achieves good efficiency with low overhead, making it a practical security enhancement.**

I. Introduction

Shor's quantum algorithm [1] is known to be able to break current traditional cryptographic algorithms' foundation, i.e., the large integer factoring and large discrete logarithm problems, given a sufficiently large quantum computer. Although such a large quantum computer has not been built so far, adoption of Post-Quantum Cryptography (PQC) is necessary to maintain the security and privacy of sensitive data in the face of the quantum computing threat. Despite progress in cryptography scheme transition such as the NIST 2024 report on Federal Information Processing Standards (FIPS) to include PQC algorithms [2], most of current applications have not adopted PQC.

Security of messaging systems has evolved in the past decades. In the networking protocol, Transport Layer Security (TLS), the fundamental encryption component in HTTPS, provides the basic data security protection for the message transmissions in the network against attacks e.g., Man-in-the-Middle attacks. In TLS 1.3, at least a 2048-bit RSA key or 256-bit ECDSA key is recommended [3]. However, some messaging applications such as Discord and Slack do not by default provide end-to-end encryptions (E2EE) for messages, which poses privacy threats to users because application provider companies can decrypt the messages stored on their servers. Among those messaging systems that have E2EE implemented, such as Whatsapp, Facebook Messenger and Telegram, most are still built only on traditional cryptographic schemes (e.g.,extended triple Diffie-Hellman [4] and ECDSA [5]) for key establishment. These messaging systems are still vulnerable to the "Harvest now, Decrypt later" threat by quantum computing based attacks.

Recent updates in Signal [6] and iMessage PQ3 [7] have adopted post-quantum cryptographic schemes to enhance security. Both have applied CRYSTALS-KYBER-1024 [8] that is based on the lattice problem difficulty [9] for key encapsulation mechanisms. Both systems are in a hybrid mode using existing cryptographic schemes such as Elliptic-curve Diffie-Hellman together with Kyber to achieve security as long as either algorithm remains unbroken. Despite the advances of these messaging systems, transitioning existing messaging applications to PQC is facing several noticeable challenges: 1) Performance and computation overhead: PQC algorithms may require more computational and communication resources for encryption, decryption, and key exchange, which requires efficient implementation of PQC especially on resource-constrained devices. 2) Cross-device support and key management: allowing users to sync conversations across multiple devices is complicated with PQC, because securely transferring quantum-resistant keys between devices while maintaining E2EE is more challenging. 3) Backward compatibility: messaging systems must maintain compatibility with existing users who may not yet support PQC.

In this paper, we present CELS, a PQC-based security extension on top of existing messaging applications, such as Discord. Like Signal and iMessage, our security extension is a hybrid approach that relies on CRYSTALS-KYBER-1024 and

This work is supported partly by grant NSF CNS 2318671.

AES-256 for key establishment. Unlike the prior works, our extension is built as a browser extension to provide end-to-end security for messaging systems. This architecture allows CELS to be applicable to different web-based messaging services. Moreover, it has the advantages of cross-device flexibility given that the PQC key is initialized by a user-controlled password. Our prototype implementation allows two communicating users to easily opt in for guarding their messages with E2EE by downloading and using the browser extension. It does not affect the users who don't adopt PQC in their communications with Discord servers. The implementation of Discord CELS using native webhooks in Firefox has shown its reasonable efficiency level through extensive experiments.

The remainder of this paper is organized as follows. After reviewing related works in Section II, we provide technical preliminary details in Section III. In Section IV, we present the details of CELS protocols. The implementation details are described in Section V. We discuss the security of CELS and the extensive experiment results on efficiency in Section VI and Section VII respectively. Finally we conclude our paper in Section VIII.

II. Related Works

A. Messaging Applications with End-to-End Encryption

Many messaging applications, such as WhatsApp [10], Signal [11], and Facebook Messenger [12], utilize the Signal protocol [13] to provide end-to-end encryption. The signal uses the extended triple Diffie-Hellman scheme to establish the secret key between two users, and the Double Ratchet algorithm based on the shared secret key can be used to exchange private messages. The Signal protocol can support both synchronous and asynchronous communications. There are other messaging applications that use proprietary E2EE protocols to provide secret chat functions, such as Telegram [14] and Element (based on Matrix) [15], where MTProto protocol [16] and Olm encryption library are used respectively [17]. A key encapsulation mechanism is often utilized to perform a key exchange and establish a shared secret. This shared secret is procedurally used with a symmetric encryption algorithm, most commonly AES-256. RSA-2048 provides 112 bits of security strength, and ECDSA-256 provides 128 bits, both in contrast to AES-256, which provides 256 bits. Although the communication that takes place after the key exchange is secured by AES-256, because that key is commonly propagated by RSA-2048, the actual security strength for the communication channel is only 112 bits, not meeting the modern minimum standard of 128 bits on top of its quantum vulnerability.

Most of the aforementioned protocols are based on RSA or Elliptic Curve Cryptography (ECC) for encryption, which are vulnerable to Shor's algorithm running on quantum computers by efficiently solving large integer factoring and discrete logarithm problems. There are a few recent advances in implementing PQC in messaging systems such as Signal with PQXDH [6] and iMessage PQ3 [7], which we will discuss in the next subsection in greater detail together with other PQC applications.

In some popular messaging applications, such as Discord and Slack, the messages are not end-to-end encrypted. Although the messages are encrypted during transmission by the TLS protocol, the companies can decrypt the messages stored on the servers and view the content. Our work provides end-to-end encryption to web-based messaging systems based on PQC in the form of browser extensions. Compared with existing E2EE solutions, it is extended directly to web-based platforms, which has better accessibility, bypassing the need for dedicated apps. At the same time, our work is quantum resistant, superior to other encryption extensions for Discord such as [18].

B. Application of Post-Quantum Cryptography in Secure Messaging

While the PQC standardization process at NIST is still ongoing, some research papers have studied PQC deployment issues in existing applications especially the impact of efficiency on embedded devices [19] and mobile networks [20]. Existing works suggest that although PQC encryption schemes have the expected computation overhead with long keys, it is feasible to deploy them in existing network and system architectures, especially in a hybrid approach when used together with traditional modern cryptographic schemes [21]–[23].

Signal has adopted PQXDH (or "Post-Quantum Extended Diffie-Hellman") Key Agreement Protocol, which provides post-quantum forward secrecy in the asynchronous setting based on the post-quantum key encapsulation mechanism CRYSTALS-KYBER-1024. In 2024, Apple released PQ3 in iMessage, a post-quantum cryptographic protocol for end-to-end secure messaging that provides not only PQC key establishment, but also ongoing PQC rekeying with a self-healing property from key compromise. There are two encryption keys used in PQ3, i.e, a Kyber-1024 public key for key encapsulation and a classical P-256 Elliptic Curve public key for key agreement. Both PQXDH and PQ3 still rely on traditional cryptographic schemes, for example ECDSA for user authentication. In our work, we also only focus on applying PQC for key encapsulation to guarantee end-to-end security and rely on existing authentication schemes in the Discord application. Apple also plans to implement PQC authentication, which is also a beneficial addition. [7] Discord CELS currently relies on Discord's native login system for authentication and non-repudiation.

To the best of our knowledge, there are no prior works that build security extensions with PQC independently on top of existing messaging systems in the web environment, which can be extended to different web-browsers and messaging systems.

III. Technical Preliminaries

In this section, we review the cryptographic preliminaries and notations used in CELS.

A. Learning With Errors

The security of the PQC algorithm that we use in CELS is due to the computational difficulty of the Module Learning with Errors (LWE) problem [24]. The LWE problem is a foundational problem in modern cryptography and complexity theory. It can be viewed as a noisy generalization of a system of linear equations over a finite field (usually Z_q), where the presence of small "errors" or "noise" makes solving the system—and thus recovering the secret—computationally difficult. Module-LWE, as a generalization of the standard LWE problem, is also believed to be hard both classically and quantumly.

B. PQC Key-Encapsulation Mechanism

A PQC key-encapsulation mechanism includes a shared key space κ, key generation algorithm, a key encapsulation algorithm and a key de-capsulation algorithm described as follows.

- $KeyGen() \to (sk, pk)$ is a probabilistic key generation algorithm that produces a secret key sk and a public key pk.
- $Enc(pk) \to (ct, ss)$ is a probabilistic key encapsulation algorithm that takes public key pk as input, and produces a ciphertext ct and a shared secret ss as output. In encryption function, a random number r is generated and used.
- $Dec(sk, ct) \to ss$ is a probabilistic key de-capsulation algorithm that takes the security key sk and ciphertext ct as input, and outputs shared secret ss.

In CELS, we use ML(Module Lattice)-KEM-1024 [25] as the PQC key encapsulation mechanism.

C. CRYSTALS-Kyber

CRYSTALS-Kyber is a post-quantum key encapsulation designed to provide secure key exchange in the presence of quantum-capable adversaries. It has been selected as a key encapsulation mechanism (KEM) (i.e., ML-KEM) in Federal Information Processing Standards (FIPS) 203 by the Secretary of Commerce [25].

Kyber as the KEM has the aforementioned components. Internally, Kyber works with polynomials in a ring (e.g., $Z_q[x]/(x^n + 1)$ for specific n) but structured as modules of rank k. The operations on polynomials enable fast implementations and also keep the keys and ciphertexts relatively short. Kyber typically comes in three security levels: Kyber-512 (targeting 128-bit security), Kyber-768 (targeting 192-bit security) and Kyber-1024 (targeting 256-bit security). These parameter sets differ in polynomial ring dimension, rank, and noise distribution parameters, balancing security and performance.

Kyber is proven IND-CCA2 secure, meaning it provides strong security guarantees even when adversaries can submit chosen ciphertexts to a decryption oracle. It also compares favorably to other lattice-based schemes in terms of runtime speed (key generation, encapsulation, decapsulation) and bandwidth (key and ciphertext sizes) [8].

IV. CELS PROTOCOLS

In this section, we introduce two CELS protocols, namely, the key establishment protocol and key retrieval, providing E2EE with postquantum security applicable to existing unprotected messaging applications.

A. Overview

CELS protocols are designed to securely establish an agreed key between two communicating parties. The postquantum cryptographic scheme CRYSTALS-Kyber-1024 is used for the key establishment and retrieval process. The key retrieval protocol provides better efficiency for parties who communicate frequently than establishing a new shared key in each new conversation. For better security, a new shared secret key can be obtained by rekeying using the key establishment protocol. The shared secret key can be used to encrypt and decrypt sensitive data between the two parties. We assume that each party has their own password to access the CELS service to initiate the key establishment or messaging process. For the same user, no keys in CELS protocols are shared across different devices/browsers.

The CELS protocols are applicable on top of messaging systems to provide quantum-attack-resistant end-to-end data security. We assume that the underlying messaging system, such as Discord, has their own user identity management, authentication, and security mechanisms, but does not provide end-to-end encryption for users. The CELS protocols focus on end-to-end key establishment and retrieval resistant to quantum attacks between two existing users of the messaging system, while leaving the user authentication to the messaging systems.

B. Key Establishment Protocol

The CELS key establishment protocol between two parties is shown in Protocol 1. The goal is to establish a secret key between Alice and Bob.

Protocol 1 CELS Key Establishment Protocol

Inputs. Alice: User-selected password P_A. Bob: User-selected password P_B.
Goal. Generate and securely share a secret key ss for the encryption of messages between two parties Alice and Bob.

The protocol:

1) Alice: $KeyGen() \to (sk_{Kyber}, pk_{Kyber})$.
2) Alice: PBKDF2(P_A)$\to k_{AES}$.
3) Alice: $Enc_{AES}(k_{AES}, sk_{Kyber}) \to k'$ and saves k' for future sessions.
4) Alice $\xrightarrow{pk_{Kyber}}$ Bob
5) Bob: $Enc_{kyber}(pk_{Kyber}) \to (ss, ct)$
6) Bob: PBKDF2(P_B)$\to k'_{AES}$.
7) Bob: $Enc_{AES}(k'_{AES}, ss) \to k''$, and save k'' for future sessions.
8) Bob $\xrightarrow{ct}$ Alice, and Alice saves ct for future sessions.
9) Alice: $Dec_{Kyber}(sk_{Kyber}, ct) \to ss$.

The main idea is that Alice first generates a Kyber public-and-private-key pair, and sends the public key to Bob. Alice also generates her own AES key based on her user-selected password P_A, which is used to encrypt the Kyber private key for her future key retrieval. Bob computes the secret key ss with Kyber encapsulation function using the public Kyber key received, and sends the ciphertext ct to Alice. Alice decapsulates ct to obtain the shared key ss.

Bob also generates his own AES key to encrypt ss for his future key retrieval. The protocol is sequential and there is Kyber public key and a ciphertext transmitted between Alice and Bob in the key establishment process (steps 4 and 8). In Protocol 1, three pieces of ciphertext (k', k'', ct) are saved for future sessions. Users can send these ciphertexts as messages to the other party to save them on the server of messaging systems.

C. Key Retrieval Protocol

To access the previously established shared secret ss in a new session, we have CELS key retrieval protocol to recover the same ss securely at the two parties using the saved information k', k'', ct in Protocol 1, and the user-selected passwords P_A and P_B.

Protocol 2 describes the process at two parties Alice and Bob respectively to retrieve the shared secret key. In particular, Alice first decrypts k' using the same AES key to recover her Kyber private key sk_{Kyber}. Then she decapsulates ct using sk_{Kyber} to obtain ss. Bob simply uses his AES key (generated by P_B) to decrypt k'' in order to retrieve ss. Compared with the key establishment protocol, key retrieval protocol does not have message exchange between the two parties, and has fewer cryptographic functions executed. Therefore, it is suitable for users who initiate conversations frequently in separate sessions, but the amount of sensitive messages exchanged is not great.

Protocol 2 CELS Key Retrieval and Messaging Protocol

Inputs. Alice: User-selected password P_A, k', ct. Bob: User-selected password P_B, k''.
Goal. Securely compute the data encryption key ss from past sessions to encrypt and decrypt messages between the two parties Alice and Bob.

Alice:
1) PBKDF2(P_A)$\rightarrow k_{AES}$.
2) $Dec_{AES}(k_{AES}, k') \rightarrow sk_{Kyber}$.
3) $Dec_{Kyber}(sk_{Kyber}, ct) \rightarrow ss$.
4) Alice uses ss to encrypt and decrypt the messages between the two parties.

Bob:
1) Bob: PBKDF2(P_B)$\rightarrow k'_{AES}$.
2) Bob: $Dec_{AES}(k'_{AES}, k'') \rightarrow ss$.
3) Bob uses ss to encrypt and decrypt the messages between the two parties.

V. Implementation

We implement the Discord CELS based on the protocols described above as a Firefox extension to provide end-to-end encrypted communication within Discord, leveraging post-quantum security (Kyber) to securely establish an AES key for actual message encryption.

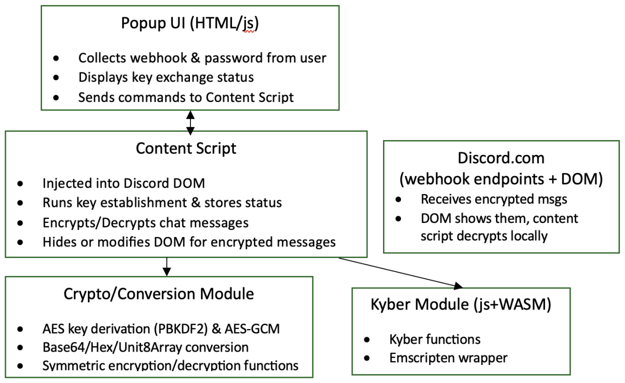
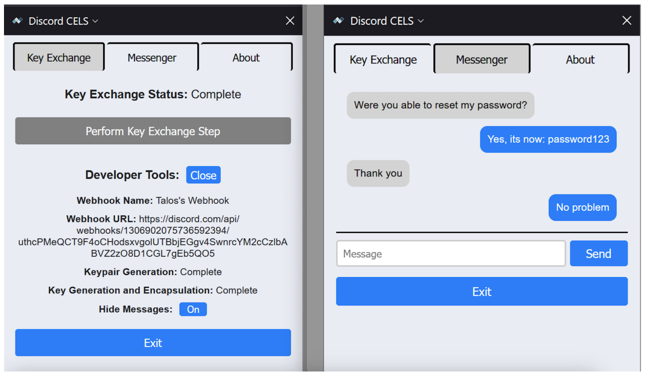

Popup UI (HTML/js)
Collects webhook & password from user
Displays key exchange status
Sends commands to Content Script
Content Script
Injected into Discord DOM
Runs key establishment & stores status
Encrypts/Decrypts chat messages
Hides or modifies DOM for encrypted messages
Discord.com
(webhook endpoints + DOM)
Receives encrypted msgs
DOM shows them, content script decrypts locally
Crypto/Conversion Module
AES key derivation (PBKDF2) & AES-GCM
Base64/Hex/Unit8Array conversion
Symmetric encryption/decryption functions
Kyber Module (js+WASM)
Kyber functions
Emscripten wrapper

Fig. 1: Discord CELS Architecture

Fig. 1 illustrates the overall architecture of our Discord CELS implementation. There are five main components in our implementation, i.e., 1) Popup UI that gathers user info and triggers key exchange steps; 2) Content script which is injected into Discord, detects new messages in real time and handles encryption/decryption automatically via DOM, manages keys and updates DOM; 3) Kyber WASM a postquantum Key Encapsulation Mechanism (KEM) compiled via Emscripten; 4) AES helpers for symmetric encryption and utility conversions and Discord + Webhook, the actual platform where messages are sent/received in encrypted form.

This structure ensures no external server sees decrypted messages; only participants with the correct password and keys can read or write cleartext.



Key Exchange Status: Complete

Developer Tools:

Webhook Name: Talos's Webhook

Webhook URL: https://discord.com/api/webhooks/1306902075736592394/uthcPMeQCT9F4oCHodsxvgolUTBbjEGgv4SwnrcYM2cCzlbABVZ2zO8D1CGL7gEb5QO5

Keypair Generation: Complete

Key Generation and Encapsulation: Complete

Hide Messages: On

Were you able to reset my password?

Yes, its now: password123

Thank you

No problem

Fig. 2: Sample screenshots of Discord CELS as a browser sidebar

Fig. 2 shows two sample screenshots of Discord CELS, for key exchange and messaging respectively. The implementation is using a small set of native browser APIs such as Web Crypto

API and Emscripten-generated code for Kyber. The source code of the CELS project is available upon request.

VI. Security Analysis

A. Threat Model

Like the analysis of security messaging systems e.g., in [17], [26], we assume three types of attackers, i.e., local adversaries (who can access the local network devices such as Wi-Fi routers on either end of the communication) and service provider adversaries, e.g., Discord or who can access Discord servers. we assume that attackers can create Discord accounts and send messages as legitimate users. The security discussions in this section are under these threat models unless otherwise stated.

B. Security Discussions

In Discord CELS, we utilize Kyber-1024 and AES-256. Kyber-1024 parameter set has been proven to achieve more than 128 bits of security and is resistant to all known classical and quantum attacks [8]. AES-256 also guarantees 128 bits of security which is mandated by NIST for all cryptographic algorithms from 2031 onward [27].

Discord CELS also uses a salt with PBKDF2 (Password-Based Key Derivation Function 2) [28]. To allow the user to securely encrypt, store, then later access their shared AES key and/or Kyber private key across distinct sessions without a central database, a 256 bit salt is used when generating key material for a user specific AES key. This allows for a rainbow table attack on commonly used passwords, but can be effectively mitigated if the user selects a password with at least 256 bits of entropy. For security configurations of PBKDF2 we use a work factor of 600,000 as recommended by OWASP [29], as well as HMAC-SHA-512 for the internal hash function. SHA-512 provides 256 bits of security strength against classical algorithms, as well as 128 bits when taking into account Grover's quantum algorithm quadratic speedup, which is still sufficiently secure [27].

The user's password entered in their credential input is stored in the browser's local storage, and its security and access control are ensured by Firefox. This allows Discord CELS to securely access the user's password for automatic message decryption, but presents possible vulnerabilities involving its persistent storage across sessions. For this reason we recommend users to press the ”Exit” button to clear Discord CELS' local storage in the browser after sessions.

Standard memory management and safety are provided by the Web Crypto API in JavaScript, and the specific Kyber C code implementation also properly frees memory after its use, both natively and through Emscripten [30].

VII. Efficiency Experiments

In this section, we perform extensive experiments on the efficiency of our Discord CELS prototype system, by examining the detailed step breakdowns of key encapsulation mechanism based on CRYSTALS-Kyber-1024, measuring the message encryption and decryption time, and evaluating the overall system overhead on top of default Discord functions.

A. Experiment Setup

Our experiments are conducted using a Desktop PC with an AMD Ryzen 5 3600XT 6-Core Processor @ 3.80GHz and Dual G.Skill 8GB RAM (F4-3600C18D-16GVK) for a total of 16GB of memory. It runs Windows 11 version 22H2 with Firefox version 133.0 and LibreWolf version 134.0-1.

In our experiments, the passwords are generated by KeepassXC 2.7.9 [31]. When the password length is fixed, a 15-character random password from the 95 ASCII characters is used due to the recommendation in [32]. In the experiments where the message length is fixed, a random 18-word message from the 999 most common English words [33] is used. This message length selection is due to a recent survey [34].

B. Key Establishment Time Breakdown

We first measure how much time each step takes in the key agreement process, i.e., key pair generation, key encapsulation and key decapsulation to understand the computation time breakdown. These Kyber local efficiency test results are the average times of 100 tests. Each test uses a random password of 15 characters long. Firefox is used for these computation time breakdown experiments.

TABLE I: Kyber Function Average Execution Time

Keypair Generation	1.52 ms
256b Random Number Generation and Encapsulation	0.88 ms
Decapsulation	0.90 ms

As we can see in Table I, the key generation takes 1.52 ms, more than the other two functions. The time it takes to generate a 256-bit random number and complete the encapsulation is 0.88ms on average. The decapsulation function takes similar amount of time 0.90ms.

C. Encryption and Decryption Time

We measure the time to encrypt the messages of various lengths. In particular, we randomly generate random messages of 18, 100 and 1000 words respectively, and test the AES Encryption and Decryption time on average of 100 tests. We also measure the time to generate the AES key based on a random password of 15 characters long. The results are shown in Table II.

TABLE II: Message Encryption and Decryption Times with Different Message Lengths

Random Message Length	PBKDF2 Key Generation Time (ms)	AES Encryption Time (ms)	AES Decryption Time (ms)
18	698.84	0.07	0.15
100	698.86	0.12	0.08
1000	695.95	0.25	0.11

We can observe that overall the AES encryption and decryption times are less than or equal to 0.25ms for different message lengths. The PBKDF2 key generation for wrapping the Kyber private key is relatively time-consuming compared to AES operations.

We also evaluate the impact of user-selected password length on message encryption and decryption time and PBKDF2 key generation time. Table III shows the results with random passwords in lengths of 15, 32, 64, 128, 256, 512, 1024, and 2048, when a fixed message length of 18 words is used. Across all password lengths, the times to encrypt and decrypt messages remain low (from 0.04ms to 0.15ms). The time for PBKDF2 key generation is in the range of 698.52ms to 701.10ms.

TABLE III: Symmetric Function Averages with Variable Password Length

Random Password Character Count	PBKDF2 Key Generation Time (ms)	AES Encryption Time (ms)	AES Decryption Time (ms)
15	698.84	0.07	0.15
32	698.65	0.06	0.04
64	700.30	0.09	0.12
128	699.31	0.07	0.04
256	701.10	0.09	0.05
512	698.52	0.11	0.05
1024	699.03	0.05	0.05
2048	699.17	0.04	0.14

D. Overhead

We finally perform the overhead test, where we run our Discord CELS on two browsers, i.e., one Firefox and one LibreWolf. One browser is acting as ”Alice” and the other as ”Bob”. Both browsers can access the Internet with a network download speed of 441.69 Mbps and an upload speed of 513.07 Mbps. Both users are using passwords with 258.50 bits of entropy, 20 word passphrases to log into the Discord CELS.

We measure the time elapse between the moment ”Alice” sends the message (clicks the button) and the moment ”Bob” receives/reads the content of the message. We perform the tests both on Discord CELS and on web-based Discord application with native integrated webhooks but without E2EE for comparison, and compute the overhead. In each comparison test, users send the same randomly generated 18-word messages, and scripts are used to read Discord's HTML DOM and record times. The comparison tests are performed ten times and the measured times are reported in Table IV.

TABLE IV: Average Overhead of Discord CELS to Deliver a 18-word Message

Discord with Webhook without E2EE	291.30 ms
Discord CELS	343.50 ms

Discord without E2EE takes 291.30ms while Discord CELS takes 343.50ms, resulting in an overhead of 17.92%, which is reasonable.

VIII. Conclusion

In this paper, we present CELS, a security extension of messaging applications to provide end-to-end encryption using post-quantum cryptography. Our prototype system Discord CELS is implemented as a Firefox sidebar extension on the web-based Discord messaging application, providing message encryption based on CRYSTALS-Kyber-1024 for communications between users. CELS is resistant to attacks launched by both classical and quantum computers, and is also flexible for existing messaging application users. Our extensive experiments have shown its efficiency to be deployed on top of Discord.

References

[1] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, Ieee, 1994.

[2] D. Moody, R. Perlner, A. Regenscheid, A. Robinson, and D. Cooper, “Transition to post-quantum cryptography standards,” tech. rep., National Institute of Standards and Technology, 2024.

[3] E. Rescorla, “The transport layer security (tls) protocol version 1.3,” tech. rep., 2018.

[4] M. Steiner, G. Tsudik, and M. Waidner, “Diffie-hellman key distribution extended to group communication,” in *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pp. 31–37, 1996.

[5] D. Johnson, A. Menezes, and S. Vanstone, “The elliptic curve digital signature algorithm (ecdsa),” *International journal of information security*, vol. 1, pp. 36–63, 2001.

[6] S. with PQXDH. https://signal.org/docs/specifications/pqxdh/.

[7] A. iMessage PQ3. https://security.apple.com/blog/imessage-pq3/.

[8] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, “Crystals-kyber: a cca-secure module-lattice-based kem,” in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 353–367, IEEE, 2018.

[9] D. Micciancio and S. Goldwasser, *Complexity of lattice problems: a cryptographic perspective*, vol. 671. Springer Science & Business Media, 2002.

[10] WhatsApp. https://www.whatsapp.com/.

[11] Signal. https://signal.org/.

[12] F. Messenger. https://www.messenger.com/.

[13] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, “A formal security analysis of the signal messaging protocol,” *Journal of Cryptology*, vol. 33, pp. 1914–1983, 2020.

[14] Telegram. https://telegram.org.

[15] Element. https://element.io.

[16] T. S. Chats. https://core.telegram.org/api/end-to-end.

[17] M. Alatawi and N. Saxena, “Sok: An analysis of end-to-end encryption and authentication ceremonies in secure messaging systems,” in *Proceedings of the 16th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, pp. 187–201, 2023.

[18] D. 2.0. https://github.com/dozer133/discrypt.

[19] P. Muzikant and J. Willemson, “Deploying post-quantum algorithms in existing applications and embedded devices,” in *International Conference on Ubiquitous Security*, pp. 147–162, Springer, 2023.

[20] J. P. Mattsson, B. Smeets, and E. Thormarker, “Quantum technology and its impact on security in mobile networks,” *Ericsson Technology Review*, vol. 2021, no. 12, pp. 2–12, 2021.

[21] S. P. Kaveri, I. Radhakrishnan, and P. B. Honnavalli, “Secure messaging in post quantum cryptography using ntruencryption,” in *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pp. 1–5, 2024.

[22] C. Döberl, W. Eibner, S. Gärtner, M. Kos, F. Kutschera, and S. Ramacher, “Quantum-resistant end-to-end secure messaging and email communication,” in *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[23] F. R. Ghashghaei, Y. Ahmed, N. Elmrabit, and M. Yousefi, “Enhancing the security of classical communication with post-quantum authenticated-encryption schemes for the quantum key distribution,” *Computers*, vol. 13, no. 7, 2024.

[24] O. Regev, “On lattices, learning with errors, random linear codes, and cryptography,” *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[25] National Institute of Standards and Technology, “Module-lattice-based key-encapsulation mechanism standard.” https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.203.pdf, 2024.

[26] N. Unger, S. Dechand, J. Bonneau, S. Fahl, H. Perl, I. Goldberg, and M. Smith, “Sok: secure messaging,” in *2015 IEEE Symposium on Security and Privacy*, pp. 232–249, IEEE, 2015.

[27] National Institute of Standards and Technology, “Recommendation for key management.” https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf, 2020.

[28] F. F. Yao and Y. L. Yin, “Design and analysis of password-based key derivation functions,” in *Topics in Cryptology–CT-RSA 2005: The Cryptographers’ Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005. Proceedings*, pp. 245–261, Springer, 2005.

[29] OWASP, “Password storage cheat sheet.” https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html, 2023.

[30] A. Zakai, “Emscripten: an llvm-to-javascript compiler,” in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pp. 301–312, 2011.

[31] KeePassXC Cross-platform Password Manager. https://keepassxc.org/, 2024.

[32] National Institute of Standards and Technology, “Digital identity guidelines.” https://doi.org/10.6028/NIST.SP.800-63b.

[33] The 1000 most common words in English. https://word-lists.com/word-lists/the-1000-most-common-words-in-english/.

[34] TextAnyWhere, “how-many-words-do-we-text.” https://www.textanywhere.com/resources/blog/technology/how-many-words-do-we-text-in-a-lifetime/, 2021.