1. Introduction

This document describes the “CLCKD” (or “Consensus-Protocol Ledger-Based CRYSTALS-Kyber/Dilithium”) key agreement protocol. CLCKD allows forward secrecy and secure signatures using CRYSTALS-Dilithium signatures[1] and CRYSTALS-Kyber post-quantum key encapsulation.[2] CLCKD provides a distributed ledger of authenticated ephemeral public keys allowing two parties to mutually authenticate each other based on constantly updating one-time public keys maintained on an XMSS distributed ledger.[3] CLCKD provides post-quantum forward secrecy, while adding a post-quantum secure signature scheme for authentication and non-repudiation where both parties are using CLCKD, or interoperability and post-quantum forward secrecy without authentication and non-repudiation when only the recipient is using CLCKD.

CLCKD is designed for asynchronous settings where one user (“Bob”) is offline but has published some information to a distributed environment. Another user (“Alice”) wants to use that

---

[1] CRYSTALS-Dilithium is lattice-based signature scheme using the hardness of Module-SIS and Module-LWE that offers speed and size advantages over hash-based schemes. The National Institute of Standards and Technology (NIST) tested and approved CRYSTALS-Dilithium as a quantum-resistant signature scheme (https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms). For further information on CRYSTALS-Dilithium, see: https://pq-crystals.org/dilithium/index.shtml.

[2] CRYSTALS-Kyber is an IND-CCA2-secure key encapsulation mechanism (KEM), whose security is based on the hardness of solving the learning-with-errors (LWE) problem over module lattices. The National Institute of Standards and Technology (NIST) tested and approved CRYSTALS-Kyber as a quantum-resistant encryption scheme (https://www.nist.gov/news-events/news/2022/07/nist-announces-first-four-quantum-resistant-cryptographic-algorithms). For further information on CRYSTALS-Kyber, see: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf

[3] XMSS, also known as RFC 8391, is publicly explained at: https://www.rfc-editor.org/info/rfc8391. The National Institute of Standards and Technology (NIST) approved RFC 8391 in Special Publication 800-208. XMSS is considered quantum-secure by NIST; the conditions for this consideration are discussed in the original research paper above.

information to send encrypted data to Bob, using a one-time public key recorded to a constantly updating and verifying ledger.

2. Preliminaries

2.1. CLCKD parameters

An application using CLCKD must decide on several parameters:

Name	Definition
HashTree	WOTS+ key pairs combined under a single public key (currently, XMSS, which is RFC 8391 compliant)
hash	A lattice-based Fiat-Shamir signature scheme (currently, CRYSTALS Dilithium, which is FIPS-204 compliant)
pqkem	An IND-CCA post-quantum key encapsulation mechanism (currently, CRYSTALS-Kyber-1024, which is FIPS-203 compliant)
pqae	A scheme for authenticated encryption that has IND-CPA and INT-CTXT post-quantum security

dag	A Directed Acyclic Graph that ties ephemeral pqkem public keys to a root user whose identity is proven yet obfuscated using zero-knowledge proofs (currently, SABRkey)
consenuspro	A decentralized protocol for verifying the integrity and non-repudiation of pqkem (currently, SABR-PAXOS)
EncodeKEM	A function that encodes a pqkem public key into a byte sequence
DecodeKEM	A function that decodes a byte sequence into a pqkem public key and is the inverse of EncodeKEM

2.2. Cryptographic notation

Throughout this document, all public keys have a corresponding private key, but to simplify descriptions we will identify key pairs by the public key and assume that the corresponding private key can be accessed by the key owner.

This document will use the following notation:

- The concatenation of byte sequences **X** and **Y** is **X** ll **Y**.
- **PQ(PK1, PK2)** represents a byte sequence which is the shared secret output from a post-quantum hash tree involving the key pairs represented by public keys PK1 and PK2. The post-quantum hash tree will be XMSS.

- **Sig(PK, M, Z)** represents the byte sequence that is an XMSS signature on the byte sequence M which was created by signing M with PK's corresponding private key. This signature verifies with public key PK and the public key is recorded to distributed ledger Z. The signing and verification methods and security of XMSS are well-documented in RFC 8391.[4]
- **IK(a, b)** represents the identity keys for ledger users. These identity keys create the ledger root for each user, but aside from the on-ledger identification they have no association with PQ or SIG.

2.3. Roles

The CLCKD protocol involves three parties: **Alice** (sender), **Bob** (receiver), and a **Distributed Ledger**.

- **Alice** wants to send **Bob** some initial data using encryption.
- **Bob** wants to allow parties like **Alice** to send him encrypted data. However, **Bob** might be offline when **Alice** attempts to do this. To enable this, **Bob** has a relationship with a **Distributed Ledger**.
- The **Distributed Ledger** stores pointers to distributed file shards that are non-computationally derived using an algorithm with information theoretic security[5] from **Alice** to **Bob** which **Bob** can later retrieve. The **Distributed Ledger** also lets **Bob** publish some data which the distributed ledger

---

[4] RFC 8391 is publicly available at: https://www.rfc-editor.org/info/rfc8391. National Institute of Standards and Technology approved RFC 8391 in Special Publication 800-208.

[5] The mathematical proofs for the data-at-rest information theoretic security algorithm are outside the scope of this paper, which is focused on data-in-transit protocols. For the mathematical proof of the data-at-rest protocol, see the SECURA paper at https://www.beskarinc.com/research/.

will provide to parties like **Alice**. The distributed ledger is inherently untrusted, but PQ and SIG are continuously and rapidly validated using a decentralized protocol for verifying integrity and non-repudiation (parameter: **consensuspro**; see 4.7 below).

2.4. XMSS

CLCKD records the following information to the Hash Tree, eXtended Merkle Signature Scheme (XMSS):

Name	Definition
EK_B	Bob’s ephemeral public key
KD_B	The disposition marker for ephemeral key EK_B

Bob has a signed ephemeral key EK_B, which is automatically updated each time that a message is received by Bob. For each signed ephemeral and disposable key, K, that Bob generates, he also computes a disposition marker, KD_B, that uniquely identifies this key on Bob’s device and to any sender. These keys and distribution markers will be uploaded to the **distributed ledger** as described in Section 3.2.

During each protocol run, Alice retrieves and a new ephemeral public key for Bob. Once the protocol is run, the disposition marker for KD_B is marked as expired and the key is no longer considered valid by consensuspro.

2.5. Post-Quantum Key Encapsulation Keys

CLCKD uses the following post-quantum key encapsulation public keys:

Name	Definition
$(PQOPK_{B}^{1}, PQOPK_{B}^{2}, \ldots)$	Bob's set of signed one-time pqkem keys

The post-quantum key encapsulation mechanism's (parameter: **pqkem**) public keys used within a CLCKD protocol run must all use the same pqkem parameter.

For each communication round, Bob generates a one-time pqkem $(PQOPK_{B}^{1}, PQOPK_{B}^{2}, \ldots)$ which is tied to Bob's directed acyclic graph root identity key (IK_B), using zero-knowledge proofs, and which is used in a single CLCKD protocol run. These keys and their corresponding identifiers will be uploaded to the **distributed ledger** as described in Section 3.2.

3. The CLCKD protocol

3.1. Overview

CLCKD has four phases:

1. **Publication**: Bob publishes his ephemeral public key and key distribution marker to a distributed ledger, recorded in XMSS.

2. **Fetching**: Alice fetches the most current public key from the distributed ledger and uses it to send an initial message to Bob.
3. **Key Expiration**: Consensuspro marks the used public key as expired and prompts Bob to generate a new public key.
4. **Key Updating**: Bob receives and processes Alice's initial message and generates a new public key and key distribution marker.

The following sections explain these phases.

3.2. Publishing keys

Bob generates and publishes a key package to a distributed ledger containing:

- Bob's obfuscated identification key
- Bob's one-time public key, EncodeKEM
- The key identifier for the one-time public key, KD_B

Bob will upload a new public key and key identifier after receiving a communication. Bob will discard the private key after a new key pair and key identifier have been identified, storing the contents of the message using a data-at-rest protocol.[6]

3.3. Sending the initial message

To perform a CLCKD key agreement with Bob, Alice contacts the distributed ledger and fetches the most current public key for Bob containing the following values

---

[6] See https://www.beskarinc.com/research/ for the mathematical proof of SECURA, one example of a data-at-rest protocol.

- Bob's ephemeral public key (EK_B)
- The one-time disposition marker for EK_B, KD_B

The ledger will provide the most current one-time public key for Bob, prompt Bob's account to automatically generate a new one-time key pair, and alert the ledger to the fact that EK_B is no longer a valid public key.

Alice verifies the disposition marker on the public key. If the disposition marker fails to validate, the protocol aborts the communication. Otherwise, if the disposition marker checks pass, Alice generates an encapsulated shared secret using our post-quantum key encapsulation mechanism (parameter: **pqkem**):

$(CT, SS) = PQKEM\text{-}ENC(PQPK_B)$
shared secret SS
ciphertext CT

If both sender and recipient are using the CLCKD protocol, Alice creates hash PH by hashing the contents of the message and signing the message using a post-quantum signature scheme, CRYSTALS Dilithium. Although XMSS could accept a classical signature scheme, XMSS is inherently post-quantum. Therefore, post-quantum signatures are used for interoperability. Furthermore, if the message needs to be verified, or the sender authenticated outside of the environment containing the distributed ledger, post-quantum signatures ensure interoperability.

Alice then sends Bob an initial message containing:

- The pqkem ciphertext CT encapsulating SS for $PQPK_B$
- The identifier, KD_B, stating which one-time key Alice used

- An initial ciphertext encrypted with a PQAE encryption scheme.
- If Alice is also using CLCKD, a CRYSTALS Dilithium hash and signature verifying the authenticity of Alice and the non-repudiation of the message.

The initial message must be encoded in an unambiguous format to avoid confusion of the message items by the recipient.

After sending this, the CLCKD protocol deletes the ciphertext CT and message hash PH.

3.4. Receiving the initial message

Upon receiving Alice's message, Bob retrieves the disposition marker from the message, indicating the corresponding private key that Bob should use to decrypt the message.

Using these keys, Bob calculates PQKEM-DEC($PQPK_B$, CT) as the shared secret SS.

Bob also generates and publishes a new key package as described in Section 3.2

4. Security considerations

CLCKD has been formally analyzed in the symbolic model with ProVerif and in the computational model with CryptoVerif. With ProVerif, the authors prove both authentication and secrecy in the symbolic model and enumerate the precise conditions under which the attacker can break these properties. These security properties notably imply forward secrecy, resistance to harvest

now decrypt later attacks, resistance to key compromise impersonation, and session independence.

Using the CryptoVerif prover, the authors prove the computational secrecy and authentication of any completed key exchange for XMSS hashes, the hash function modeled as a random oracle, and the IND-CPA+INT-CTXT assumptions for the PQAE. Moreover, they also show forward secrecy when the signature was UF-CMA secure at the time the key exchange took place, assuming post-quantum IND-CCA security for the KEM, modeling the hash function as a PRF, and IND-CPA+INT-CTXT security for the PQAE.

The remainder of this section discusses a list of known security considerations.

4.1. Authentication and Non-Repudiation

A message sent using the CLCKD protocol where both Alice and Bob communicate using the CLCKD communication protocol, authenticates the communication by appending to each communication a lattice-based Fiat-Shamir hash and signature developed from IK_A or IK_B.[7] This method allows for non-repudiation of the communication. When only one party to the communication, Alice or Bob, sends a communication via the SABRkey communication API, the communication loses authentication and non-repudiation. If authentication is not performed, the parties receive no cryptographic guarantee as to

[7] Fiat-Shamir heuristics are beyond the scope of this paper, but are available in the original proposal at https://link.springer.com/chapter/10.1007/3-540-47721-7_12. The original proposal failed to include a proof of security; Pointcheval and Stern later provided this proof, given a random oracle, at: https://link.springer.com/chapter/10.1007/3-540-68339-9_33

with whom they are communicating. The methods by which authentication and non-repudiation are applied are discussed in Section 3.3.

Authentication in CLCKD is only quantum-secure when both the sender and the receiver use the CLCKD communication API or a similar post-quantum authentication protocol. CLCKD, through decentralized ledger entries, is designed to be interoperable with any other post-quantum system using Crystals-Kyber KEM, such as PQ3/PQ4.

4.2. Protocol replay

It is possible that an adversary could send multiple encrypted messages to Bob before KD_B and EK_B are updated on the distributed ledger. This replay attack will be accepted, causing Bob to think Alice had sent him the same message (or messages) repeatedly.

To mitigate this, the distributed ledger uses a patented fast-consensus decentralized protocol. The details of this fast-consensus protocol are beyond the scope of this paper.[8] In testing, the fast-consensus protocol processed over 1,000,000 communications per second using commonly-available computer hardware. The speed and scalability of the ledger in real-world scenarios remains unknown, but is assumed to be slower than the testing environment due to competing server processor demands, bandwidth delays, and simultaneous memory requests. The specific details of the ledger and its computational testing results are beyond the scope of this document but research results are

[8] For further information on the SABR-PAXOS fast-consensus protocol, see https://www.beskarinc.com/research/.

publicly available in *Quantum-Secured Decentralized Directed Acyclic Graphs*.

4.3. Deniability

CLCKD intentionally does not provide cryptographic deniability *in the face of a third party with access to a communication party's secret keys*. While a third party cannot determine from ciphertext that Alice and Bob communicated, and cannot determine the contents of their conversation, all communications within CLCKD are purposefully hashed and signed to provide authentication and non-repudiation (see 4.1 above). However, to provide authentication and non-repudiation, a third party still needs access to plaintext, which requires that the third party have access to one or more party's secret keys and is presented with a transcript created by communication between Alice and Bob. In the absence of this scenario (i.e., the third party has access to neither Alice's nor Bob's secret keys), offline deniability is accomplished.

Similar to other post-quantum communication protocols, we focus on offline deniability because if either party is collaborating with a third party during protocol execution, they will be able to provide proof of their communication to such a third party.

In the absence of a cooperating party or a third party having access to Alice or Bob's secret keys, the deniability of communications is assumed to be limited by the underlying hardness of the Pedersen commitment. While discrete logarithms are assumed to be classically secure, they are also assumed to be breakable through quantum computation.[9] In CLCKD, the post-

[9] See https://arxiv.org/pdf/2307.03065.pdf for the susceptibility of discrete logarithms to quantum computational decryption.

quantum value v of C = vG + rH, where G and H are known generator points on an elliptic curve, creates a level of post-quantum deniability as there is no known algorithm that can determine the plaintext version of v.[10] However, we assume that an attacker using a post-quantum computer could determine the ciphertext of v (V_c) and then compare that value to all other V_c on a distributed ledger. An uncomputable but non-zero level of deniability is still available to the user based on the fact that computing V_c, even with a quantum computer, is computationally impractical.[11]

4.4. Key compromise

Unique to CLCKD, compromise of a private key only affects a single communication round (or trivial set of rounds) between Alice and Bob. The specific methods by which these private keys are unlocked depends on the authentication protocol of the underlying system using the CLCKD protocol.[12]

4.5. Passive quantum adversaries

CLCKD is designed to prevent “harvest now, decrypt later” attacks by adversaries with access to a quantum computer capable of computing discrete logarithms in curve. While this security is primarily derived from pqkem, it also requires that pqae provides post-quantum IND-CPA and INT-CTXT security. Unique to

---

[10] V is computationally determined by pqkem.

[11] CLCKD uses a bit size exceeding the ability of the fastest known classical processor on Earth (Frontier, at 1.2 exaFLOPS) to resolve a secret (theoretically assuming a similar device was capable of processing in qubits, which Frontier can not) for the remaining lifetime of the Earth.

[12] For further reading on one such system, see the IDENTICO authentication protocol, available at https://www.beskarinc.com/products/.

CLCKD, the rapid replacement of one-time key pairs allows a unique level of protection from passive quantum adversaries without the need for additional considerations.

4.6. Active quantum adversaries

CLCKD is designed to provide protection against active quantum attackers through one-time keys authentication of each message (see 4.1 above). An active attacker with access to a quantum computer capable of computing discrete logarithms in curve could not overcome the authentication of the message.

4.7. Distributed Ledger Trust

Although the specifics of the directed acyclic graph used by CLCKD and its related fast-consensus protocol are beyond the scope of this paper, the ledger is purposefully designed to be untrusted. However, the consensus results are provably trusted based on proposal u being the value with the highest proposal number among all responses that process P receives from the processes of accepting a strict majority of proposals m and u.[13]

4.8. Risks of weak randomness sources

The security of PQKEM relies on the random sources available when running the **PQKEM-ENC** operation. If a user has weak entropy then the resulting PQKEM will have low entropy. To mitigate a weak randomness issue, we recommend a

[13] For further information, see the mathematical proof of SABR-PAXOS, available at https://www.beskarinc.com/research/.

cryptographically secure pseudorandom number generator seeded with an external source of entropy such as /dev/urandom in Linux distributions.

4.9 Preventing KEM Re-encapsulation Attacks

Kyber KEM incorporates the KEM public key into the generation of the shared secret, preventing KEM Re-encapsulation attacks. Any derivation from FIPS-103 is assumed to no longer be compatible with the CLCKD protocol.

5. IPR

6. Acknowledgements

The CLCKD protocol was developed by Daniel Chapple.

Thank you to Nicholas Grokhowsky, a doctoral candidate at North Carolina State University's Center for Geospatial Analytics, for his editorial feedback.

Thanks to the Kyber team for their work on the Kyber key encapsulation mechanism.