TRANSITIONING TO POST-QUANTUM KEY MANAGEMENT FRAMEWORK

TECHNICAL REPORT

Mahabir Prasad Jhanwar[*1], **Aarav Varshney**[† 1], **and Adit Dhawan**[‡ 1]
[1]Department of Computer Science, Ashoka University, Sonipat, India

September 20, 2023

ABSTRACT

With the advent of quantum computing on the horizon, there is an urgent need to transition cryptographic systems to post-quantum algorithms. This paper describes the implementation of a post-quantum key management framework (KMF) that allows secure key exchange and authentication between a key distribution center (KDC) and a communicating entity (Lab). To achieve post-quantum confidentiality, we utilize PQ Crystals's Kyber key encapsulation mechanism (KEM) for quantum-safe key exchange and modify it to encapsulate larger keys. For authentication, we leverage recent work on post-quantum X.509 certificates and generate certificates signed using PQ Crystals's Dilithium digital signature algorithm. This provides a blueprint for transitioning KMFs to a post-quantum setting with confidentiality and authentication guarantees against quantum attacks.

1 Introduction

Cryptographic techniques rely on cryptographic keys that are managed and protected throughout their lifecycle by a key management framework (KMF). Effective cryptography reduces the need to protect large amounts of data down to just protecting the keys.

However, recent advances in quantum computing will likely lead to large-scale quantum computers that can break widely used public key cryptosystems like RSA via Shor's algorithm. This makes classical KMF implementations vulnerable for high-security use cases like governments. To address this looming threat, we need to transition to post-quantum cryptographic algorithms.

In this report, we describe a post-quantum KMF that allows a key distribution center (KDC) to generate confidential material and transfer it securely to a communicating entity (X) with post-quantum security guarantees. There are two main challenges in designing such a framework: ensuring confidentiality and authentication in a post-quantum setting.

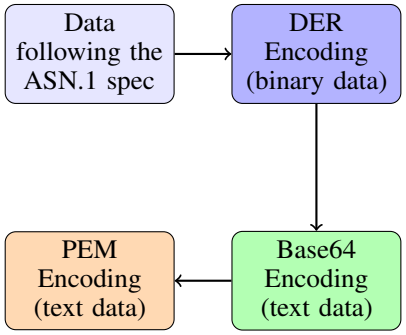
Confidentiality can be achieved with symmetric encryption like AES, which is secure against quantum attacks. However, the key exchange protocol establishing a shared secret between the KDC and Lab must also be quantum-safe. We use PQ Crystals's Kyber key encapsulation mechanism (KEM) for this and modify it to encapsulate keys larger than 32 bytes. Authentication typically relies on long-term public keys stored in X.509 certificates. To enable post-quantum authentication, we leverage recent work on post-quantum X.509 certificates and design certificates signed using PQ Crystals's Dilithium digital signature scheme.

[*]mahavir.jhawar@ashoka.edu.in
[†]aarav.varshney@ashoka.edu.in
[‡]adit.dhawan_asp24@ashoka.edu.in

```
Certificate ::= SEQUENCE {
 tbsCertificate ToBeSigned,
 signatureAlgorithm AlgorithmIdentifier,
 signature BIT STRING }

ToBeSigned ::= SEQUENCE {
 version [0] Version DEFAULT v1,
 serialNumber SerialNumber,
 signature AlgorithmIdentifier,
 issuer Name,
 validity Validity,
 subject Name,
 subjectPublicKeyInfo SubjectPublicKeyInfo,
 issuerUniqueID [1] IMPLICIT UID OPTIONAL,
 subjectUniqueID [2] IMPLICIT UID OPTIONAL,
 extensions [3] Extensions OPTIONAL }
```

(a) ASN.1 structure of X.509 certificate



(b) Steps in building an X.509 certificate.

Figure 1: Overall components involved in X.509 certificate generation.

2 Design

2.1 Key Setup

The KDC generates long-term key pairs for signing and key exchange with X. Specifically, it generates a Dilithium public/private key pair ($\mathsf{pk}^{\mathsf{s}}_{\mathsf{KDC}}$, $\mathsf{sk}^{\mathsf{s}}_{\mathsf{KDC}}$) for signing and a Kyber KEM public/private key pair ($\mathsf{pk}^{\mathsf{e}}_{\mathsf{X}}$, $\mathsf{sk}_{\mathsf{X}}{}^{e}$) for encrypted key exchange. To complete the initial setup, the KDC sends $\mathsf{pk}^{\mathsf{s}}_{\mathsf{KDC}}$, $\mathsf{pk}^{\mathsf{e}}_{\mathsf{X}}$ and $\mathsf{sk}_{\mathsf{X}}{}^{e}$ to X. We implement this in two phases: first building Python wrappers around the Dilithium and Kyber C source code libraries, and then using these wrappers to generate the key pairs in Python. Next, we leverage our certificate generation code to create two X.509 certificates: a self-signed certificate for the KDC containing $\mathsf{pk}^{\mathsf{s}}_{\mathsf{KDC}}$, and a KDC-signed certificate for X containing both $\mathsf{pk}^{\mathsf{e}}_{\mathsf{X}}$ and $\mathsf{sk}_{\mathsf{X}}{}^{e}$ to enable authenticated key exchange.

Dilithium and Kyber. We build a wrapper around Dilithium by compiling the Dilithium source code from `https://github.com/pq-crystals/dilithium` into a shared library (.so on Linux, .dll on Windows). We then load this shared library in Python and use the ctypes module to call the underlying C functions for Dilithium from our Python code. Specifically, we export three methods from the wrapper: `keygen`, `sign`, and `verify`. `keygen` generates a Dilithium public/private key pair. `sign` takes a message and private key as input and returns a signature. `verify` takes a message, signature, and public key as input and returns a boolean indicating if the signature is valid. We use the same overall process to build a Python wrapper around Kyber as well. Abstracting Dilithium and Kyber into Python wrappers allows us to leverage these C-based libraries in our project cleanly from Python code.

X.509 Certificates. An X.509 certificate binds a public key to the identity of the owner of the matching private key (the subject), and identifies the entity that vouches for this binding (the issuer). Certificates also contain validity periods, extensions for revocation checking, and usage constraints. We construct a certificate using the ASN.1 structure defined in Boeyen et al. [2008]. The Abstract Syntax Notation One (ASN.1) is a language for describing structured data like certificates. As shown in Figure 1, given an ASN.1 specification and corresponding values, encoding rules derive a binary representation of the certificate. Although many encoding rules exist, X.509 certificates use Distinguished Encoding Rules (DER) which ensures unique serialization - no two distinct certificates have the same encoding. Furthermore, we use PEM encoding to convert the binary DER-encoded string into a readable text format. PEM encoding base64-encodes the data and adds header/footer lines to identify the content type.

The ASN.1 schema for X.509 certificates relies on two key data structures - SubjectPublicKeyInfo and AlgorithmIdentifier:

```
SubjectPublicKeyInfo ::= SEQUENCE {
        algorithm AlgorithmIdentifier,
        subjectPublicKey BIT STRING
}
```

```
AlgorithmIdentifier ::= SEQUENCE {
        algorithm OBJECT IDENTIFIER,
        parameters ANY DEFINED BY algorithm OPTIONAL
 }
```

The subjectPublicKey field contains the public key, while algorithm identifies the algorithm via an object identifier (OID).

To integrate Dilithium signatures, we modify the ASN.1 schema as follows:

First, in the Certificate sequence definition, we update two fields:

- The signatureAlgorithm field is changed to use the Dilithium object identifier (OID), specifically 1.3.6.1.4.1.2.267.7.4.4 from the OQS OpenSSL fork Open Quantum Safe Project.
- The signature value is updated to contain a Dilithium signature.

In the ToBeSigned sequence:

- The signature field is same as the updated signatureAlgorithm in the Certificate sequence.
- The subjectPublicKeyInfo field is of type SubjectPublicKeyInfo. Within this type, the algorithm field is changed to the Dilithium OID and subjectPublicKey contains a Dilithium public key.

In summary, we update the OID fields and algorithm-specific values to leverage Dilithium signatures in X.509 certificates. The modifications use ASN.1's agility to enable post-quantum algorithms.

2.2 Key Transport

The KDC uses pk_X^e to encapsulate a shared secret key k_s and sends the encapsulated key $\mathsf{c}_{\mathsf{k}_s}$ to X. The KDC uses its private key $\mathsf{sk}_X{}^e$ to retrieve k_s.

References

Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008. URL `https://www.rfc-editor.org/info/rfc5280`.

Open Quantum Safe Project. Open quantum safe openssl. `https://github.com/open-quantum-safe/openssl`. Accessed: September 20, 2023.