Provably Secure Butterfly Key Expansion from the CRYSTALS Post-Quantum Schemes

Edward Eaton[1], Philippe Lamontagne[1,2] and Peter Matsakis[1]

[1] National Research Council Canada
[2] Université de Montréal, Montréal, Canada

Abstract. This work presents the first provably secure protocol for Butterfly Key Expansion (BKE) – a tripartite protocol for provisioning users with pseudonymous certificates – based on post-quantum cryptographic schemes. Our work builds upon the CRYSTALS family of post-quantum algorithms that have been selected for standardization by NIST. We extend those schemes by imbuing them with the additional functionality of *public key expansion*: a process by which pseudonymous public keys can be derived by a single public key. Our work is the most detailed analysis yet of BKE: we formally define desired properties of BKE – unforgeability and unlinkability – as cryptographic games, and prove that BKE implemented with our modified CRYSTALS schemes satisfy those properties. We implemented our scheme by modifying the Kyber and Dilithium algorithms from the LibOQS project, and we report on our parameter choices and the performance of the schemes.

Keywords: Pseudonymous Certificates · Kyber · Dilithium · Post-Quantum

1 Introduction

Public key digital signatures are a powerful and convenient tool to establish the authenticity of communications in a digital setting. Digital signatures are used to sign web pages, emails and programs and form an important line of defense against many classes of cyberattacks, such as phishing, man-in-the-middle, malware and supply chain attacks. Public keys are the identity users employ to verify signatures. A valid signature for a message under a particular public key gives confidence that the holder of the corresponding private key did indeed sign the message.

In practice, public keys are not useful identifiers when compared to names, email addresses and URLs. So keys are bundled with other identification information. This bundle is then signed by a central authority to form a *certificate*. Anyone who trusts the *certificate authority* (CA) can validate the identity of certificate holders by storing only the CA's public key.

In addition to authenticity, public key signatures offer non-repudiability. If pk is associated with Alice's identity through a valid CA certificate, any valid signature σ for a message m can be traced back to her and her secret key sk. Moreover, she cannot claim to not have signed m since only she should be able to produce valid signatures for pk. This is a problem when privacy or anonymity is desired. For example, one may consider a network where exchanges are authenticated (only authorized entities may send valid messages), but anonymous (messages can't be linked to a given entity). To achieve this, the CA could issue many pseudonymous certificates to each entity, each with a different public key. Then Alice can use her public key a few times before rotating to a new key. This has a big communication overhead if keys are rotated often, since Alice must generate and send many public keys to the CA to be signed. A clever trick to save on bandwidth is to send a single public key and have the CA *expand* the key into multiple new public keys. If the signature scheme has some *homomorphic* property between its private and public keys, then Alice can derive the corresponding private key for each expanded public key. For example in ECDSA, key pairs are of the form $\mathsf{sk} = u$ and $\mathsf{pk} = u \cdot G$ where G is a curve point. The CA expands pk by adding a new multiple of G, obtaining $\mathsf{pk}' = u \cdot G + u' \cdot G$ and signs pk'. If the CA sends pk' and u' to Alice, she can compute $u + u'$ as the corresponding secret key.

A downside to the above approach is that the CA is now trusted for both the authenticity and the anonymity of the scheme. This is remediated with the introduction of a new entity, the registration authority (RA), responsible for anonymizing requests sent to the CA, such that no single entity can break anonymity. Briefly, the RA amasses many public keys, does a first round of key expansion and shuffles the keys before sending them to the CA. The CA expands them again and signs the resulting keys sending them back to the RA, who forwards them to Alice. To prevemt the RA from linking expanded keys to identities, Alice also sends the RA her public key for an encryption scheme, which is expanded by the RA and used by the CA to encrypt the certificates. The resulting scheme is called[1] *butterfly key expansion* (BKE) [Why+13].

1.1 Related Work

Butterfly key expansion as a solution to the pseudonymity problem in certificate provisioning comes from a Security Credential Management System (SCMS) designed for vehicle-to-vehicle (V2V) communication [Why+13]. It is one of the leading candidates for credential management in V2V, with a recent proof of concept by CAMP [LLC16] and receiving scrutiny by the US Department of Transportation [Uni19] and Transport Canada [Can19]. The full SCMS includes other entities such as a Linkage Authority and a Misbehavior Authority which as the names imply exist to identify (link) and revoke certificates issued to misbehaving entities. Since our work focuses on unlinkability and unforgeability of certifiate issuance, we do not include those authorities which deal with revocation. A subsequent work [Sim+18] improves the bandwidth of the SCMS by using the same public key for both the signature and encryption schemes.

In our literature review, we found two papers focused on implementing butterfly key expansion with post-quantum schemes.

The first is a 2018 preprint [Bar+18] that focuses on building a *unified* butterfly key expansion protocol. In unified butterfly key expansion [Sim+18], only one public key is sent and expanded, which is used for both signing and encryption. For discrete-logarithm based systems this is relatively straightforward, but becomes more challenging for lattice-based schemes. Even though public keys in many lattice-based protocols are module learning with errors samples, the parameters, rings, and distributions of values are tailored much more specifically than in discrete logarithm systems.

In [Bar+18], the authors based their signature scheme on qTESLA [Alk+20] (a Ring-LWE signature scheme conceptually similar to Dilithium that was not selected for standardization by the NIST process due to efficiency concerns). The key exchange protocol was based on the Lyubashevsky-Peikert-Regev system [LPR13], which is also part of the lineage of Kyber.

In a more recent paper [SB23] the authors report the results of replacing elliptic curve cryptography with the post-quantum system NTRU [Che+20] in butterfly key expansion. As noted in previous work [Bar+18], the fundamental property that enables butterfly key expansion is a homomorphism between the secret key and public key domains, something which both LWE and discrete logarithm based solution enjoy. NTRU has no obvious

[1]The *caterpillar key of Alice is expanded into* cocoons *by the RA and expanded again into* butterflies *by the CA.*

homomorphic structure between its secret and public key domains, making it unclear precisely how butterfly key expansion with NTRU might operate.

The method by which public keys are expanded in BKE is similar to the notion of public key blinding in the context of digital signatures, which rerandomizes the public key in such a way that it is unlinkable to the original public key and the secret key owner can still issue valid signatures with respect to the blinded public key. Post-quantum key blinding schemes (including for Dilithium) were first introduced in [ESS21].

1.2 Our Contributions

We introduce variants of the Kyber and Dilithium encryption and signature schemes that allow for their public keys to be expanded – a means by which public keys can be rerandomized while preserving functionality. We present methods for expanding public keys and to rederive the corresponding secret keys such that the scheme is still secure with respect to expanded keys. When we instantiate the butterfly key expansion protocol with our variants, we obtain a certificate management system that improves upon previous works in a few key ways.

Standardized Schemes Our protocol is based on Dilithium and Kyber, which have been selected for standardization by NIST. Our protocol makes minimal changes, largely focusing on small parameter tweaks. In this way, our protocol benefits in part from the public scrutiny Dilithium and Kyber have received over the years, as well as the strong performance characteristics of the two. The end result is smaller as well, with the final encrypted package sent to vehicles being around 2000 bytes smaller than [Bar+18][2].

Provable Security There are often significant gaps in the scrutiny received by cryptographic primitives compared to more complex cryptographic protocols which are often presented without security reductions. Our work closes this gap for butterfly key expansion: we introduce a framework for provable security of BKE. We define the notions of unlinkability and unforgeability of BKE as cryptographic games. We prove that our scheme satisfies those definition assuming the hardness of MLWE, in the random oracle model. Our definitions are general enough to account for subtle attacks that an adversary could potentially launch, like being able to see which expanded public keys a vehicle uses or doesn't use, and whether that leaks identifying information. This level of detail in the security analysis was not present in prior works.

Implementations and Benchmarki We have implemented the tweaked Kyber and Dilithium schemes as a fork of the widely used PQC library LibOQS [SM17][3]. We report on the performance of our implementation in Appendix 5.

Unified Butterfly Key Expansion Previous works introduced the *unified* variant of BKE, in which the expanded public keys are used for both encryption and signature verification. Reusing keys for different purposes is generally seen as unsafe (see for example Appendix A.1). We describe what types of signature and encryption schemes can be safely combined to use the same public key. Our results show that Dilithium and Kyber public keys can in theory be safely used for both encryption and signature verification, as long as there exist sets of parameters where the public keys of both schemes are compatible.

[2]See Appendix 5.

[3]`https://github.com/open-quantum-safe/liboqs`

2 Background

2.1 Notation and Preliminaries

For $n \in \mathbb{N}$, we let $[n] = \{0, \ldots, n-1\}$. For a set S, we let $u \leftarrow\$ S$ denote that u is sampled uniformly at random from S. We write $X \leftarrow D$ to denote that X is sampled according to distribution D. Kyber and Dilithium use the following distributions. We let B_η as the binomial distribution of width η: sample $(a_1, \ldots, a_\eta, b_1, \ldots, b_\eta) \in \{0,1\}^{2\eta}$ uniformly at random and output $X = \sum_{i=1}^{\eta}(a_i - b_i)$. We let S_η denote the uniform distribution over $[-\eta, \eta]$.

The hardness assumption on which both Kyber and Dilithium rely is the *module learning with errors* assumption.

Definition 1 (Module learning with errors with static A matrix)**.** Let R_q be the ring $\mathbb{Z}_q[X]/(X^n+1)$ of polynomials over $\mathbb{Z}_q$ and let D be a probability distribution over R_q. Let $A \leftarrow\$ R_q^{m\times k}$, let $u_1, \ldots, u_N \leftarrow\$ R_q^m$ and let $s_1, \ldots, s_N \leftarrow D^k$ and $e_1, \ldots, e_N \leftarrow D^m$. The decisional module learning with errors problem with static A matrix (S-MLWE) consists of distinguishing $(A, u_1, \ldots, u_N)$ from $(A, As_1+e_1, \ldots, As_N+e_N)$. For any PPT algorithm $\mathcal{A}$, we define $\mathsf{Adv}^{\text{S-MLWE}}_{\mathcal{A},D}$ as the advantage of adversary $\mathcal{A}$ in distinguishing both distributions.

The regular MLWE assumption corresponds to $N = 1$ in the above definition. The distribution D is typically the discrete Gaussian distribution. Kyber and Dilithium use distributions B_η and S_η respectively, as defined above. We state the S-MLWE *assumption* as the assumption that $\mathsf{Adv}^{\text{S-MLWE}}_{\mathcal{A},D} \leq \mathsf{negl}(n)$ for $D \in \{B_\eta, S_\eta\}$, where the exact values for η can be found in Sections 3 and 4. We observe that if the MLWE problem is hard with distribution D, then so is the S-MLWE problem, which reuses the same A matrix over multiple samples. A standard hybrid argument (to our knowledge first used in the context of LWE in [PW11]) can reduce S-MLWE to LWE. The proof is straightfoward, so refer to Appendix B for the details.

2.2 Butterfly Key Expansion

Butterfly key expansion (BKE) was introduced in [Why+13] in the context of vehicle-to-vehicle (V2V) communications. The BKE process provisions a vehicle with an arbitrary number of pseudonymous certificates while satisfying two fundamental properties:

- The computational and communication burden on the vehicle is minimized. In particular, its role is limited to providing two public keys and receiving the certified public keys.
- Neither the Registration Authority nor the Certificate Authority alone is capable of compromising the pseudonimity of issued certificates.

Butterfly key expansion works as follows: A single pair of *caterpillar* public keys (one for encryption, one for signing)[4] is generated by the vehicle and provided to the Registration Authority (RA). The RA takes these keys and checks the vehicle's eligibility to receive a collection of pseudonymous certificates. After this check, the caterpillar public keys $(\mathsf{pk}_s, \mathsf{pk}_e)$ for a signature and encryption scheme are expanded into many public keys $\{(\mathsf{pk}_{s,i}, \mathsf{pk}_{e,i})\}_{i\in[N]}$. The RA does this by pseudorandomly generating additional public keys with a seed known to the vehicle and adding them to the caterpillar keys. As noted in [Bar+18], the essential property to enable this is a homomorphism between the private and public key spaces. The result of adding together two public keys is still a public key,

[4]In the *unified* variant of BKE, the same public key is used for both the signature and encryption schemes.

with the secret key corresponding to the sum of the two individual secret keys.[5] The intent of this process is that the caterpillar public keys are entirely re-randomized, such that observing the resulting public keys reveals no information about the original, but such that the original caterpillar secret keys are still required to sign or decrypt messages. The RA is then meant to produce and collect a large number of these keys, from a large number of vehicles. The expanded public keys from many vehicles are then 'shuffled' to provide unlinkability against the CA. This bundle of shuffled expanded key pairs is forwarded on to the certificate authority.

The certificate authority (CA) receives this bundle of public key pairs, and is unable to link together any two pairs as coming from the same vehicle. The CA expands each signing key once more to ensure unlikability against the RA. The CA then creates a certificate for the doubly expanded signing keys. Note that the RA has never seen this public key, and the certificate authority has no way to connect one pseudonymous public key to another. The remaining task is to get the certificates and the final public keys back to the vehicle. The CA uses the expanded encryption key to encrypt the certificate and its secret expansion nonce. The CA signs and sends the resulting ciphertexts back to the RA, so that the RA can sort the encrypted packages out and forward them to the correct vehicle.

Finally, the vehicle receives a collection of encrypted and signed packages $\{pkg_i\}_{i\in[N]}$ from the RA. Now, whenever they need a new pseudonymous key pair and certificate they derive the expanded encryption secret key, decrypt the package, and then derive the (doubly expanded) signature key pair along with the CA's certificate for this key pair.

This process is illustrated in Figure 1. For the purpose of simplicity, this figure omits the important step of having the RA bundle and shuffle together expanded public keys from many vehicles. This is the step that provides unlinkability for the vehicle against the CA.

In order to be used in a BKE scheme, the public key encryption and signature schemes must be augmented with the following procedures.

- $\mathsf{Expand}(\mathsf{pk}, \tau)$ produces an expanded public key pk' using the random seed τ. For MLWE–based schemes, this means sampling a new public key and adding it to pk.
- $\mathsf{Receive}(\mathsf{sk}, \tau)$ computes the secret key sk' corresponding to the expanded key pk' derived from pk and τ.

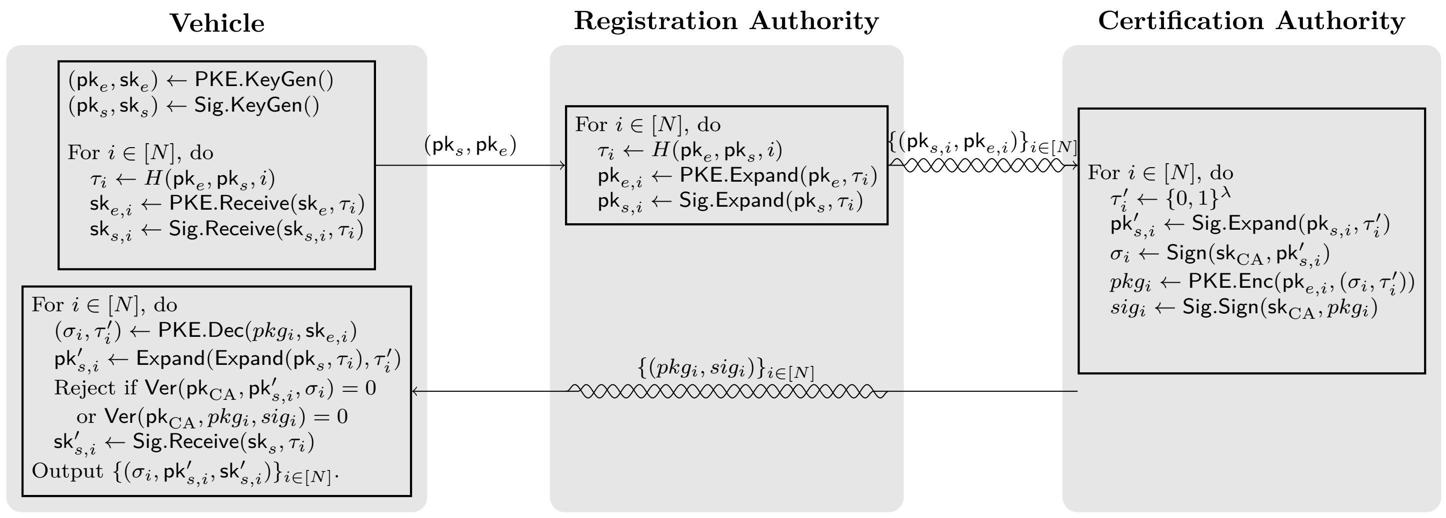
The security properties that we ask of such public-key schemes are the same as for the original, but they should hold with respect to the expanded key pair $(\mathsf{pk}', \mathsf{sk}')$ for any seed τ.

The Expand algorithm is explicit in our modifications to Kyber and Dilithium, but we do not implement the Receive procedure directly in Kyber. Instead, since encryption keys are only used once to decrypt the packages, secret key reception for Kyber is done in the decryption procedure, which receives an additional parameter τ. For Dilithium, public keys are expanded twice, so the Receive procedure receives two expansion seeds τ and τ' and computes the secret key for the doubly expanded public key.

2.3 Security Model

The two stated goals of security with respect to butterfly key expansion are that of unlinkability and unforgeability. Unlinkability refers to the scheme's effectiveness at providing the expected level of pseudonymity. In other words, the resulting public keys and certificates (as well as any signatures issued under those public keys) cannot be 'linked' as having originated from the same vehicle, except possibly through the use of out-of-band

[5]For MLWE based schemes, the public keys are of the form $As + e$ with private key s. Adding two public keys (with the same A matrix) yields $(As + e) + (As' + e') = A(s + s') + (e + e')$ which has the associated private key $s + s'$.



Vehicle
Registration Authority
Certification Authority
$(\mathsf{pk}_s, \mathsf{pk}_e)$
$\{(\mathsf{pk}_{s,i}, \mathsf{pk}_{e,i})\}_{i\in[N]}$
$\{(pkg_i, sig_i)\}_{i\in[N]}$

```
$(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{PKE.KeyGen}()$
$(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{Sig.KeyGen}()$

For $i \in [N]$, do
    $\tau_i \leftarrow H(\mathsf{pk}_e, \mathsf{pk}_s, i)$
    $\mathsf{sk}_{e,i} \leftarrow \mathsf{PKE.Receive}(\mathsf{sk}_e, \tau_i)$
    $\mathsf{sk}_{s,i} \leftarrow \mathsf{Sig.Receive}(\mathsf{sk}_{s,i}, \tau_i)$
```

```
For $i \in [N]$, do
    $\tau_i \leftarrow H(\mathsf{pk}_e, \mathsf{pk}_s, i)$
    $\mathsf{pk}_{e,i} \leftarrow \mathsf{PKE.Expand}(\mathsf{pk}_e, \tau_i)$
    $\mathsf{pk}_{s,i} \leftarrow \mathsf{Sig.Expand}(\mathsf{pk}_s, \tau_i)$
```

```
For $i \in [N]$, do
    $\tau_i' \leftarrow \{0,1\}^\lambda$
    $\mathsf{pk}'_{s,i} \leftarrow \mathsf{Sig.Expand}(\mathsf{pk}_{s,i}, \tau_i')$
    $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_{\mathrm{CA}}, \mathsf{pk}'_{s,i})$
    $pkg_i \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{e,i}, (\sigma_i, \tau_i'))$
    $sig_i \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}_{\mathrm{CA}}, pkg_i)$
```

```
For $i \in [N]$, do
    $(\sigma_i, \tau_i') \leftarrow \mathsf{PKE.Dec}(pkg_i, \mathsf{sk}_{e,i})$
    $\mathsf{pk}'_{s,i} \leftarrow \mathsf{Expand}(\mathsf{Expand}(\mathsf{pk}_s, \tau_i), \tau_i')$
    Reject if $\mathsf{Ver}(\mathsf{pk}_{\mathrm{CA}}, \mathsf{pk}'_{s,i}, \sigma_i) = 0$
      or $\mathsf{Ver}(\mathsf{pk}_{\mathrm{CA}}, pkg_i, sig_i) = 0$
    $\mathsf{sk}'_{s,i} \leftarrow \mathsf{Sig.Receive}(\mathsf{sk}_s, \tau_i)$
Output $\{(\sigma_i, \mathsf{pk}'_{s,i}, \mathsf{sk}'_{s,i})\}_{i\in[N]}$.
```

Figure 1: The Butterfly key expansion process with a single vehicle. The full protocol would include shuffling keys from different vehicles by the registration authority (which provides unlinkability against the certificate authority and is represented here by the twirling lines) and the inclusion of additional metadata in the certificate.

data (for example, having observed the origin of two pseudonyms as being the same vehicle). The unlinkability property is meant to hold even against the registration authority and certificate authority. It is easy to see that unlinkability cannot hold against a collaborating registration and certificate authority. We therefore can only prove unlinkability by making a non-collusion assumption between the RA and the CA.

Unforgeability refers to the idea that only the vehicle should be able to produce signatures for their public keys. Because we are concerned about the possibility of a malicious certificate authority, defining what constitutes a public key belonging to a vehicle becomes trickier. A malicious CA can clearly make their own public key, create a certificate for it as if it belongs to a vehicle, and then create signatures all without the vehicle's knowledge or interaction with the protocol. Indeed, this problem is not unique to SCMS and corrupted certificate authorities constitute a significant problem in general. For our purposes, the best we can hope for is that if the vehicle has accepted a butterfly public key as belonging to them (that is, they have reconstructed it at the end of the pseudonymous certificate provisioning process) then only they can craft signatures for such a key. Anyone else, including the CA, cannot create signatures for a public key that the vehicle might actually use. This helps to minimize the amount of trust that we place in the CA. Note however, that unforgeability does not require an assumption that the RA and the CA do not collude.

This results in essentially three security properties (and proofs) being needed – unlinkability against the CA, unlinkability against the RA, and unforgeability against the RA and CA collaborating. Note that these three models imply security in other settings, such as against an external adversary not involved in the certificate provisioning process. We now define those security properties as security games.

2.3.1 Unlinkability against a Malicious Certificate Authority

For unlinkability against a malicious certificate authority, we ask that it should not be able to tell, given two certificates it emitted, whether they correspond to the same vehicle or not. Unlinkability against the CA is enforced by the random shuffling of the expanded keys from multiple vehicles that is performed by the registration authority.

To model unlinkability against the CA, we conceive a game between the malicious CA and a challenger. The challenger plays the role of both the vehicle and the RA: it generates signature and encryption keys for two vehicles, expands them as the RA would, shuffles them and sends them to the CA. The goal of the malicious CA is now to output a pair of indices (i, j) such that the expanded public key pairs $(\mathsf{pk}_{e,i}, \mathsf{pk}_{s,i})$ and $(\mathsf{pk}_{e,j}, \mathsf{pk}_{s,j})$ originate from the same vehicle. To capture all possible strategies to the malicious CA in the packages it sends back to the RA, we give it access to a decryption oracle[6] for the expanded encryption keys and to a signature oracle for the expanded signature keys. The BKE scheme is unlinkable against a malicious CA if the probability of winning the game is not noticeably larger than the trivial $\frac{N-1}{2N-1} = 2 \cdot \binom{N}{2} / \binom{2N}{2}$.

Definition 2 (Unlinkability of BKE against malicious certificate authority)**.** We define unlinkability against a malicious certificate authority (UL-CA) of butterfly key expansion as the following game.

1. For $b \in \{0, 1\}$, the challenger samples key pairs $(\mathsf{pk}_e^{(b)}, \mathsf{sk}_e^{(b)}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and $(\mathsf{pk}_s^{(b)}, \mathsf{sk}_s^{(b)}) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$ for a public key encryption scheme Enc and signature scheme Sig.
2. For each $b \in \{0, 1\}$, the challenger computes N expansions of the public keys $(\mathsf{pk}_e^{(b)}, \mathsf{pk}_s^{(b)})$, resulting in $2N$ expanded keys $\{(\mathsf{pk}_{e,i}, \mathsf{pk}_{s,i})\}_{i\in[2N]}$
3. The challenger picks a random permutation π of $2N$ elements and sends $\{(\mathsf{pk}_{e,\pi(i)}, \mathsf{pk}_{s,\pi(i)})\}_{i\in[2N]}$ to the adversary.
4. The adversary has, for each $i \in [2N]$, oracle access to a key decapsulation oracle for $\mathsf{PKE.Dec}(\mathsf{pk}_{e,i}, \cdot)$ and to a signing oracle for $\mathsf{Sig.Sign}(\mathsf{pk}_{s,i}, \cdot)$.
5. The adversary outputs two indices $i, j \in [2N]$. The challenger outputs 1 if the key pairs indexed by i and j were both expanded from the same key pair $(\mathsf{pk}_e^{(b)}, \mathsf{pk}_s^{(b)})$ for some $b \in \{0, 1\}$.

The advantage of a malicious CA in the unlinkability game is defined as $\mathsf{Adv}^{\text{UL-CA}} = \left|\Pr[\text{Challenger outputs 1}] - \frac{N-1}{2N-1}\right|$.

2.3.2 Unlinkability against a Malicious Registration Authority

For unlinkability against a malicious registration authority, we ask that it should not be able to distinguish whether two sets of expanded public keys with associated certificates belong to the same vehicle or to different vehicles.

We model this as a game (Definition 3) between a challenger and the malicious RA where the challenger takes the role of both vehicle and certificate authority. In the game, the challenger sends a pair of public keys $(\mathsf{pk}_e, \mathsf{pk}_s)$ to the malicious RA which produces a set of expanded keys $\{(\mathsf{pk}_{e,i}, \mathsf{pk}_{s,i})\}_i$. The challenger now either expands and produces certificates for the signature keys $\mathsf{pk}_{s,i}$ produced by the RA or it samples a fresh public key pk'_s, expands it as an honest RA would as $\{\mathsf{pk}'_{s,i}\}_i$ and produce the CA message using this set of expanded keys. If the malicious RA cannot distinguish between which course of action the challenger took, it means that it cannot distinguish between certificates for pk_s and for pk'_s. The definition 3 below also contains other steps to account for the full attack surface available to the RA.

[6]In butterfly key expansion, the CA sends a ciphertext which is ultimately decrypted by the vehicle. Giving the CA query access to the decryption oracle accounts for any linking attacks that may occur this way.

Definition 3 (Unlinkability against malicious registration authority)**.** We define the unlinkability experiment against a malicious registration authority as the following game:

1. The challenger, acting as the vehicle, samples two key pairs $(\mathsf{pk}_e, \mathsf{sk}_e) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$ and $(\mathsf{pk}_s, \mathsf{sk}_s) \leftarrow \mathsf{Sig.KeyGen}(1^\lambda)$; sends pk_e and pk_s to the adversary.
2. The adversary sends N public key pairs $\{\mathsf{pk}_{e,i}, \mathsf{pk}_{s,i}\}_{i\in[N]}$ to the challenger.
3. The challenger, now acting as the certificate authority, samples a key pair $(\mathsf{pk}_{CA}, \mathsf{sk}_{CA}) \leftarrow \mathsf{Sig.KeyGen}()$ and a bit $b \in \{0, 1\}$.
   - If $b = 0$, it proceeds to the next step.
   - If $b = 1$, it samples a fresh signature key pair $(\hat{\mathsf{pk}}_s, \hat{\mathsf{sk}}_s)$ and replaces $\mathsf{pk}_{s,i}$ with $\mathsf{pk}_{s,i} \leftarrow \mathsf{Sig.Expand}(\hat{\mathsf{pk}}_s, H(\mathsf{pk}_e, \hat{\mathsf{pk}}_s, i))$ for $i \in [N]$.
4. The challenger computes the N packages and signs them as the CA would, and sends the resulting message $\{(pkg_i, sig_i)\}_i$ to the adversary (message from CA to RA).
5. The adversary sends $\{(pkg'_i, sig'_i)\}_i$ to the challenger (message from RA to vehicle).
6. The challenger checks the validity of each package:
   (a) $(\mathsf{pk}'_{s,i}, \sigma_i, \tau'_i) \leftarrow \mathsf{PKE.Dec}(pkg_i, \mathsf{sk}_{e,i})$
   (b) Abort the game if σ_i or sig_i are invalid signatures or if $\mathsf{pk}'_{s,i}$ was not expanded from pk_s (if $b = 0$) or from $\hat{\mathsf{pk}}_s$ (if $b = 1$).
7. Send the list of certificates $\{(\sigma_i, \mathsf{pk}'_{s,i})\}_i$ to the adversary.
8. The adversary outputs a guess b' for b.

The advantage of an adversary against the above game is defined as $\mathsf{Adv}^{\text{UL-RA}} = \left|\Pr[b = b'] - \frac{1}{2}\right|$.

For our BKE scheme, unlinkability against the RA is enforced through the encryption of the certificates using the Kyber expanded public keys. Since the Dilithium public keys sent from the RA to the CA are expanded again by the CA, they will look independent from the vehicle's public key by the MLWE assumption. Because those keys are encrypted with the vehicle's Kyber expanded keys before being sent to the RA, it will not be able to link an expanded Dilithium key with its certificate to the vehicle's identity public key pk_s. To show this, we prove that Kyber still has ciphertext indistinguishability with respect to expanded public keys (Theorem 2). The full proof is found in Section 6.2.

2.3.3 Unforgeability

In the context of butterfly key expansion, we ask that the signature scheme used by the vehicle remains unforgeable even against actively colluding RA and CA. Observe that we cannot hope to achieve unlinkability in this setting. We should also note that if the certificate authority wishes to usurp the identity of a vehicle, it can just sample a signature key pair and create a certificate for this key pair with the identity of the vehicle in the certificate metadata. The reason why we consider both the RA and the CA as adversaries when proving unforgeability is to show that the key expansion process does not allow forgeries, even when this expansion is performed by malicious entities.

We refer to the RA and CA as a single entity: the *adversary*. The adversary's goal is to produce a signature public key $\hat{\mathsf{pk}}_s$ along with a certificate for $\hat{\mathsf{pk}}_s$ such that 1- the vehicle accepts the public key and the certifiate and 2- the adversary is able to produce a forgery for $\hat{\mathsf{pk}}_s$. Intuitively, the adversary cannot win such a game with non-negligible probability since part of the secret key for the expanded public keys $\mathsf{pk}'_{s,i}$ remains known only to the vehicle. We define unforgeability of expanded signatures with respect to doubly expanded keys because of the two rounds of expansion that occur in BKE. This trivially implies unforgeability for a single round of expansion.

Definition 4. We define *unforgeability under chosen key expansion and signature attack* of a doubly expanded signature scheme as the following game.

1. The challenger samples a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Sig.KeyGen}()$ and sends pk to the adversary.
2. The adversary sends $2N$ seeds $\{(\tau_i, \tau_i')\}_{i\in[N]}$. For each $i \in [N]$, the challenger expands pk twice:
   (a) $\mathsf{pk}_i' \leftarrow \mathsf{Sig.Expand}(\mathsf{Sig.Expand}(\mathsf{pk}_s, \tau_i), \tau_i')$
   (b) $\mathsf{sk}_i' \leftarrow \mathsf{Sig.Receive}(\mathsf{Sig.Receive}(\mathsf{sk}_s, \tau_i), \tau_i')$
3. The adversary has oracle access to $\mathsf{Sign}(\mathsf{sk}_i', \cdot)$ for $i \in [N]$.
4. The adversary outputs a message m^*, a signature σ^* and an index i^* such that (m^*, i^*) was not previously queried to the signature oracle $\mathsf{Sign}(\mathsf{sk}_{i^*}', \cdot)$.

The advantage of an adversary $\mathcal{A}$ in this game is defined as the probability $\mathsf{Adv}_{\mathcal{A}}^{\text{UF-exCMA}} = \Pr[\mathsf{Sig.Ver}(\sigma^*, m^*, \mathsf{pk}_{i^*}') = 1]$.

The UF-exCMA security of expanded Dilithium is proved under Theorem 4 in Section 6.3. Our proof relates the unforgeability of expanded Dilithium to that of "regular" Dilithium: we construct a reduction, which uses an adversary against expanded Dilithium and tries to produce a forgery for a target Dilithium public key $\hat{\mathsf{pk}}_D$. The reduction first *guesses* for which expanded public key the adversary will produce a forgery, then reprograms the random oracle such that this expanded public key will correspond to $\hat{\mathsf{pk}}_D$. This guess will be correct with inverse polynomial probability and thus leads to a non-negligible advantage against Dilithium if the adversary against expanded Dilithium succeeds with non-negligible probability.

3 Expanding Kyber

We modify the Kyber protocols so one can add together two public keys, encrypt under the resulting public key and decrypt under the induced secret key. Pseudocode for the modified protocols are presented in Figure 2 and we briefly describe the changes here.

- $\mathsf{Kyb.KeyGen}$ is essentially identical except that every public key uses the same $\hat{A}$ matrix (i.e. it is not sampled using a seed ρ as in regular Kyber). This ensures the additive homomorphic relation between public and private keys.
- $\mathsf{Kyb.Expand}(\mathsf{pk}, \tau)$ is a new algorithm for expanding public keys using τ as a seed for the pseudorandom values. This algorithm first generates a new Kyber public key using $H(\mathsf{pk}, \tau)$ instead of the random value σ in $\mathsf{Kyb.KeyGen}$ and adds this new public key to pk (in the NTT domain) to obtain the expanded public key.
- $\mathsf{Kyb.Encaps}(\mathsf{pk})$ is left unchanged with the exception that the $\hat{A}$ matrix is fixed.
- $\mathsf{Kyb.Decaps}(c, \mathsf{sk}, \tau)$ takes an additional input the seed τ used to expand the public key. It uses τ to derive the secret key sk' for the expanded public key $\mathsf{pk}' = \mathsf{Kyb.Expand}(\mathsf{pk}, \tau)$ and then decapsulates the ciphertext in the same way as Kyber.

Only Kyber's KEM meets the stronger IND-CCA2 security definition. Since BKE needs a public key encryption (PKE) scheme, it will simplify the notation to speak of Kyber as a PKE. An IND-CCA2 KEM can be combined with an IND-CCA2 symmetric encryption scheme in a hybrid scheme, resulting in an IND-CCA2 PKE [CS03].

```
Decaps$(c, sk, \tau)$
1 :  $pk \leftarrow \mathsf{sk} + 12 \cdot k \cdot n/8$
2 :  $h \leftarrow \mathsf{sk} + 24 \cdot k \cdot n/8$
3 :  $z \leftarrow \mathsf{sk} + 24 \cdot k \cdot n/8 + 32$
4 :  $\vec{u} \leftarrow \mathsf{Decompress}_q(\mathsf{Decode}_{d_u}(c), d_u)$
5 :  $v \leftarrow \mathsf{Decompress}_q(\mathsf{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$
6 :  $\hat{s} \leftarrow \mathsf{Decode}_{12}(sk)$
7 :  $N \leftarrow 0$
8 :  foreach $i \in \{0, \ldots, k-1\}$ do
9 :      $s'[i] \leftarrow \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(\mathsf{pk}, \tau, N))$
10 : $\hat{s}' \leftarrow \mathsf{NTT}(\vec{s}')$
11 : $\hat{s}_\tau \leftarrow \hat{s} + \hat{s}'$
12 : $m' \leftarrow \mathsf{Encode}_1(\mathsf{Compress}_q(v - \mathsf{NTT}^{-1}(\hat{s}_\tau^T \circ \mathsf{NTT}(\vec{u})), 1))$
13 : $pk' \leftarrow \mathsf{Expand}(pk, \tau)$
14 : $\overline{K}', r' \leftarrow G(m' \| H(pk'))$
15 : $c' \leftarrow CPA.Enc(pk', m', r')$
16 : if $c = c'$ then
17 :     return $K \leftarrow \mathsf{KDF}(\overline{K}' \| H(c))$
18 : else
19 :     return $K \leftarrow \mathsf{KDF}(z \| H(c))$
```

```
Expand$(pk, \tau)$
1 :  $N \leftarrow 0$
2 :  foreach $i \in \{0, \ldots, k-1\}$ do
3 :      foreach $j \in \{0, \ldots, k-1\}$ do
4 :          $\hat{A}[i][j] \leftarrow \mathsf{Parse}(\mathsf{XOF}(j, i))$
5 :  foreach $i \in \{0, \ldots, k-1\}$ do
6 :      $s'[i] \leftarrow \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(H(\mathsf{pk}, \tau), N))$
7 :      $N \leftarrow N + 1$
8 :  foreach $i \in \{0, \ldots, k-1\}$ do
9 :      $e'[i] \leftarrow \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(H(\mathsf{pk}, \tau), N))$
10 :     $N \leftarrow N + 1$
11 : $\hat{s}' \leftarrow \mathsf{NTT}(\vec{s}')$
12 : $\hat{e}' \leftarrow \mathsf{NTT}(\vec{e}')$
13 : $\hat{t}' \leftarrow \hat{A} \circ \hat{s}' + \hat{e}'$
14 : $\hat{t} \leftarrow \mathsf{Decode}_{12}(pk)$
15 : $\hat{t}_\tau \leftarrow \hat{t} + \hat{t}'$
16 : $pk_\tau \leftarrow \mathsf{Encode}_{12}(\hat{t}_\tau \pmod{^+} q))$
17 : return $pk_\tau$
```

Figure 2: The expanded Kyber algorithms with changes to Decaps highlighted in red. The key generation and encapsulation algorithms are essentially unchanged with the exception that the A matrix expanded from a seed is now fixed.

Parameter	Description	Kyber768	Our System
n	Ring dimension	256	256
q	Ring modulus	3329	3329
k	Number of ring elements	3	3
η_1	Binomial width for s, e, and r	2	2 or 4
η_2	Binomial width for e_1 and e_2	2	2
d_u	Number of bits to encode coefficients of u	10	12
d_v	Number of bits to encode coefficients of v	4	6
$\lvert\mathsf{pk}\rvert$	Public key size in bytes	1184	1152
$\lvert\mathsf{ct}\rvert$	Ciphertext size in bytes	1088	1344

Figure 3: The parameters that define an instance of Kyber, and their values for the latest Kyber768 parameter set and for our system. Our system only changes the number of bits used to encode the ciphertext (u, v). Note that the size of the public key in our system is actually smaller, simply because we do not use ρ to determine the A matrix, instead having all participants in the system use the same A matrix for pseudonymity.

3.1 Parameter Selection

In Figure 3 we describe the parameters of Kyber, and what parameters we have chosen for our system. We compare this with the parameter set that our set most closely resembles, that of Kyber768. Changes are colored in red. Note that despite basing our parameters on Kyber768, we are not necessarily trying to maintain exactly the same level of security as in Kyber768. We have purposely chosen a higher parameter set (than Kyber512) in order to have a buffer to accommodate small losses in tightness or security induced by the expansion procedure.

For our purpose, changing the rings R and R_q would be undesirable, as much of Kyber's efficiency comes from the careful choice of this ring and the NTT that is possible over it. Therefore, making minimal changes means only altering k, the η values, representing the width of the binomial distribution, and the d values, the degree of rounding employed.

When two M-LWE public keys $As_1 + e_1$ and $As_2 + e_2$ are added together (which use the same "A" matrix), this is equivalent to using the secret key $(s_1 + s_2)$ with the public key $A(s_1 + s_2) + (e_1 + e_2)$. When we encrypt with respect to such a matrix, we can analyze the security by considering the resulting distribution of secret keys. Since components of s and e are drawn from B_{η_1}, the binomial distribution of width parameter η_1, it is easy to see that the distribution of $s_1 + s_2$ is $B^k_{2\cdot\eta_1}$ and the same is true for $e_1 + e_2$ (recall that additions are over the polynomial ring). In other words, using $s_1 + s_2$ as the secret key is equivalent to doubling the η_1 parameter during key generation. So for our system we sample secrets from $B^k_{\eta_1}$, but analyze security as if secrets were drawn from $B^k_{2\cdot\eta_1}$.

Doubling η_1 widens the M-LWE secret distribution, which increases the probability of a decryption failure. There is a growing body of analysis that shows that even a single decryption failure is unacceptable from a security perspective, and that therefore decryption failures must be cryptographically unlikely.

In the Kyber768 parameter set, doubling the width of the binomial s and e are drawn from (and changing no other parameters) increases the probability of a decryption failure from 2^{-165} to 2^{-83}, a dramatic and unacceptable difference. To offset this while staying with R and R_q, we can simply reduce d_u and d_v. These parameters control how much the ciphertext is rounded for increased efficiency. While they do have an impact on security (the rounding is treated as slightly increasing the size of the error distribution), their effect is minimal. By increasing d_u and d_v, we can reduce the probability of a decryption error back to a cryptographically negligible rate, while only increasing the size of the ciphertext by a moderate amount.

The proposed parameter set in Figure 3 achieves a negligible decryption error rate[7] of 2^{-146} at a cost of increasing the size of the ciphertext from 1088 bytes to 1344 bytes. The main change made in this parameter set is the reduction in the extent to which we round the ciphertext. As noted in the Kyber specification [Ava+20], Kyber is much less reliant on rounding than LWR schemes because it also adds noise. Rounding is estimated to increase security by roughly 6 bits, so rounding less does not substantially decrease the security level.

4 Expanding Dilithium

The modified Dilithium algorithms are presented in Figure 4. As in Kyber, an important change to the original Dilithium scheme is that all public keys use the same A matrix. This change gives the homomorphic property to the public keys required for BKE. We describe the most important changes below.

- $\mathsf{Dil.KeyGen}$ uses the same $\hat{A}$ matrix for every public key, so it is expanded deterministically. The helper function $\mathsf{Power2Round}$ is replaced with a new function $\mathsf{Power2RoundHigh}$ which only returns the high order bits t (instead of splitting it into high and low bits) since the lower order bits will be computed in $\mathsf{Receive}$ after the public key is expanded.

- $\mathsf{Dil.Expand}(\mathsf{pk}, \tau)$ is a new algorithm that expands the public key pk by generating a new public key using τ as seed and returns the sum of both public keys. Note that as in KeyGen, only the higher order bits of the MLWE sample are kept as part of the key.

- $\mathsf{Dil.Receive}(\mathsf{pk}, \mathsf{sk}, \tau, \tau')$ is a new algorithm that takes as input a public/private key pair and two expansion seeds, and returns the secret key corresponding to the doubly expanded public key $\mathsf{pk}' = \mathsf{Expand}(\mathsf{Expand}(\mathsf{pk}, \tau), \tau')$. When the RA and CA are expanding the public key, they don't have access to the lower order bits of the MLWE sample $t = A \cdot s_1 + s_2$ from which pk is obtained (since Dilithium public keys consist of the high order bits of t). Therefore, the expanded public key that eventually gets signed by the CA is slightly off from the "full" public key (i.e. what you would get if you added together three MLWE samples and kept only their high order bits). The t_Δ variable included in the private key serves a similar role to t_0, but it also compensates for this difference between the "full" public key and the signed public key. The fact that t_Δ is larger than t_0 is the rational for increasing ω, the maximum Hamming weight of hints, relative to the Dilithium3 standards.

- $\mathsf{Dil.Sign}(m, \mathsf{sk})$ is essentially the same algorithm as in Dilithium except that it uses a fixed A matrix and computes the hints using t_Δ.

- $\mathsf{Dil.Ver}(\mathsf{pk}, m, \sigma)$ is also essentially the same as in Dilithium except it uses a fixed A matrix.

- $\mathsf{ExpandS}(\rho)$ is a pseudorandom generator that takes as input a seed ρ and produces a pair $(s_1, s_2) \in [-\eta, \eta]^\ell \times [-\eta, \eta]^k$.

[7] As estimated by the same failure rate estimation script used by Kyber [DS21]. Note that this corresponds to the 'one-shot' decryption failure rate.

```
Dil.KeyGen()
1: $\zeta \leftarrow_{\$} \{0,1\}^{256}$
2: $(\rho', K) \in \{0,1\}^{512} \times \{0,1\}^{256} \leftarrow H(\zeta)$
3: $\hat{A} \leftarrow \mathsf{ExpandA}()$
4: $(s_1, s_2) \leftarrow \mathsf{ExpandS}(\rho')$
5: $t \leftarrow \mathsf{NTT}^{-1}(\hat{A} \cdot \mathsf{NTT}(s_1)) + s_2$
6: $t_1 \leftarrow \mathsf{Power2RoundHigh}_q(t, d)$
7: return $(\mathsf{pk} = t_1, sk = (K, s_1, s_2))$
```

```
Dil.Expand(pk, τ')
1: $\hat{A} \leftarrow \mathsf{ExpandA}()$
2: $(s'_1, s'_2) \leftarrow \mathsf{ExpandS}(\tau')$
3: $t' \leftarrow \mathsf{NTT}^{-1}(\hat{A} \cdot \mathsf{NTT}(s'_1)) + s'_2$
4: $t'_1 \leftarrow \mathsf{Power2RoundHigh}_q(t', d)$
5: $t'' \leftarrow t'_1 + \mathsf{pk}$
6: return $t''$
```

```
Dil.Receive(pk, sk = (K, s_1, s_2), τ', τ'')
1: $\hat{A} \leftarrow \mathsf{ExpandA}()$
2: $(s'_1, s'_2) \leftarrow \mathsf{ExpandS}(\tau')$
3: $(s''_1, s''_2) \leftarrow \mathsf{ExpandS}(\tau'')$
4: $\bar{s}_1 \leftarrow s_1 + s'_1 + s''_1$
5: $\bar{s}_2 \leftarrow s_2 + s'_2 + s''_2$
6: $\bar{t} \leftarrow \mathsf{NTT}^{-1}(\hat{A} \cdot \mathsf{NTT}(\bar{s}_1)) + \bar{s}_2$
7: $t_\Delta \leftarrow \bar{t} - \mathsf{pk} \cdot 2^d$
8: $tr \leftarrow H(\mathsf{pk})$
9: return $\mathsf{sk} = (K, tr, \bar{s}_1, \bar{s}_2, t_\Delta)$
```

Figure 4: The expanded Dilithium key generation, public key expansion and secret key recovery algorithms. Changes in the key generation algorithm are highlighted in red. The signature and verification algorithms are almost identical with the exception that the matrix A is fixed and that the public and secret keys used are computed using Expand and Receive.

Parameter	Description	Dilithium 2	Our System	Dilithium 3
n	Ring dimension	256	256	256
q	Ring modulus	8380417	8380417	8380417
(k, ℓ)	Dimensions of $A \in \mathcal{R}_q^{k \times \ell}$	(4,4)	(5,4)	(6,5)
τ	Hamming weight of c	39	39	49
d	Bits dropped from public key coeffs.	13	13	13
γ_1	Coefficient range for $\vec{y}$	2^{17}	2^{19}	2^{19}
γ_2	Low-order rounding range	$(q-1)/88$	$(q-1)/32$	$(q-1)/32$
η	Range for secret key	2	2 or 6	4
β	Bound on size of c times secret vector	78	234	196
ω	Maximum Hamming weight of hint	80	70	55
$\lvert\mathsf{pk}\rvert$	Public key size in bytes	1312	1600	1952
$\lvert\sigma\rvert$	Signature size in bytes	2420	2667	3293
	Expected signing repetitions	4.25	4.96	5.1

Figure 5: The parameters that define an instance of Dilithium, and their values for the latest Dilithium 2 & 3 parameter sets and for our system.

4.1 Dilithium Parameters

As in Kyber, we change as few parameters as necessary to still achieve a high level of security. Unlike in Kyber, we expand Dilithium public keys *twice* for butterfly key expansion. Also unlike Kyber, the security of Dilithium is based on both the module LWE and SIS (short integer solution) problem, so parameters must be tuned to ensure that both the underlying M-LWE and M-SIS problems remain difficult. We base our parameters off of a mixture of the Dilithium 2 and 3 parameter sets for a level of security between the two.

By adding three secrets with size at most η, we greatly reduce the probability of finding a signature in the while loop. Accordingly, we increase other parameters to keep the number of expected repetitions similar but accommodate the larger secrets.

For a list of parameters, their descriptions, and the values in Dilithium 2 and in our system we refer to Figure 5. For security, we need to ensure that the underlying LWE and SIS problems remain difficult. We rely upon the security estimation scripts used by Kyber and Dilithium to evaluate the Core SVP hardness of the underlying instances [DS21]. The underlying M-LWE instance has a block size of 422, 123 bits of classical core-SVP hardness, and 111 bits of quantum core-SVP hardness (the numbers for the M-LWE instance in Dilithium 2 are 423, 123, and 112 respectively). Note that the M-LWE instance here is taken with $\eta = 2$, as we need the M-LWE instance to be indistinguishable from uniform for a single sample in order to guarantee unlinkability.

The underlying M-SIS instance has a block size of 508, 148 bits of classical core-SVP hardness, and 134 bits of quantum core-SVP hardness (for Dilithium 2 these numbers are 423, 123, 112). Note that we have evaluated the hardness only with respect to the SIS instance induced by existential unforgeability, as our proof does not aim for strong unforgeability.

5 Implementation Details

We provide an implementation of the cryptographic primitives based on Kyber and Dilithium. Our implementation is a fork of the PQCrystals reference implementations (in C), as found in libOQS [SM17]. Our code is based on version `0.8.0-rc1` of libOQS. We used Dilithium2 and Kyber768 as the basis of our schemes and have modified their set of parameters as presented in Sections 3.1 and 4.1.

We have compared the running time of our new and modified algorithms against the reference implementations of Kyber768 and Dilithium2. Our findings, reported in Fig. 6, are consistent with what one might expect. Key expansion takes about the same time as key generation for the expanded variants of Kyber and Dilithium. For Kyber, key generation is slightly faster because of the use of a constant A matrix and decapsulation is slower since it must compute the expanded key and the corresponding secret key from the expansion seed. For expanded Dilithium, the runtimes are comparable to Dilithium2 despite the fact that our parameters sit somewhere between Dilithium2 and Dilithium3.

Bandwidth Requirements The outgoing message for the vehicle in the Butterfly Key Expansion process based on our implementation is 2752 bytes for a pair of expanded Kyber and Dilithium public keys. The incoming packages consist of (pkg, sig) where sig is a (regular) Dilithium2 signature of pkg. If we instantiate the PKE scheme with our expanded Kyber KEM and AES encryption with the resulting key, then pkg consists of an expanded Kyber ciphertext along with an AES encryption of τ' and a signature σ of the expanded public key pk'_s. With 64 bytes for τ, the complete package size is $2 \cdot |sig| + |ct| + |\tau| = 6248$ bytes.

6 Security of Butterfly Key Expansion

In this section, we define and prove the security of Butterfly Key Expansion when instantiated with expanded Kyber and Dilithium according to the security properties presented in Section 2.3.

6.1 Unlinkability Against a Malicious Certificate Authority

Our proof that a malicious CA cannot win the unlinkability game with probability much better than the trivial $\frac{N-1}{2N-1}$ in our BKE scheme relies on two main facts. First, since Kyber and Dilithium are based on the Fujisaki-Okamoto and Fiat-Shamir transforms, respectively, the signature and decryption oracle queries can be simulated without access to the secret keys by using random oracle recording and reprogramming. Second, because the public keys are expanded using random MLWE samples, by the S-MLWE assumption all keys look uniformly random to the PPT malicious RA.

Operation	Mean time (μs)	pop. stdev	Operation	Mean time (μs)	pop. stdev
Expanded Dilithium			**Dilithium2**		
keypair	96.892	13.829	keypair	88.701	13.606
expand	98.692	10.895			
receive	224.825	75.465			
sign	548.570	390.405	sign	457.450	343.217
verify	115.938	47.940	verify	94.720	8.155
Expanded Kyber			**Kyber768**		
keygen	52.773	57.852	keygen	63.325	11.820
expand	39.292	5.450			
encaps	72.623	7.401	encaps	71.063	8.814
decaps	141.257	13.690	decaps	82.750	11.147

Figure 6: Benchmarking of the protocols on an Intel Core i5-7300U 2.60GHz processor. Each protocol ran for 100 000 iterations. The Dilithium2 and Kyber768 figures are for the reference implementation for version `0.8.0-rc1` of libOQS.

Theorem 1. *The butterfly key expansion scheme instantiated with expanded Dilithium and expanded Kyber is unlinkable against malicious certificate authority (Definition 2) with advantage*

$$\mathsf{Adv}^{\text{UL-CA}} \leq 4q_{RO}2^{-h_\infty} + 2N(\epsilon_{dec} + \epsilon_{sig}) + \mathsf{Adv}^{\text{S-MLWE}}$$

where ϵ_{dec} and ϵ_{sig} are the overheads of simulating the decryption and signature oracles of Kyber and Dilithium, respectively, q_{RO} is the number of oracle queries and h_∞ is a lower-bound on the min-entropy of Kyber and Dilithium public keys.

Proof. In Definition 2, the malicious certificate authority is presented with $2N$ expanded Kyber and Dilithium public key pairs $\rho_1, \rho_2, \ldots, \rho_{2N}$ of the form $\rho_i = (\mathsf{pk}_{K,i}, \mathsf{pk}_{D,i})$ with N pairs expanded from an original key pair $\rho^{(0)} := (\mathsf{pk}_K^{(0)}, \mathsf{pk}_D^{(0)})$, and N expanded from $\rho^{(1)} := (\mathsf{pk}_K^{(1)}, \mathsf{pk}_D^{(1)})$. These $2N$ public keys are shuffled together and presented to the CA (note that $\rho^{(0)}$ and $\rho^{(1)}$ are kept hidden from the CA). The CA then has access to a decryption and signing oracle, allowing it to request any ciphertext to be decrypted with respect to any of the $2N$ Kyber public keys and any message to be signed for the $2N$ Dilithium keys. Eventually the CA must submit two indices, i and j ($i \neq j$), and is said to win if pairs ρ_i and ρ_j were either both derived from $\rho^{(0)}$ or both derived from $\rho^{(1)}$. Assuming the security of S-MLWE, we want to establish that any adversary's success is negligibly close to trivial success probability of $\frac{N-1}{2N-1}$.

We proceed by a game hopping argument in the random oracle model. Game G_0 corresponds to the original game.

In game G_1, we modify the operation of the decryption oracle so that decryption is *simulated* using the random oracle rather than by using the (expanded) secret key. This is made possible by the (slightly modified) Fujisaki-Okamoto transform that Kyber employs. Fujisaki-Okamoto is used to upgrade the security of an IND-CPA secure KEM or PKE to IND-CCA2. This is done by establishing that decryption queries against public key can be handled without requiring the secret key, which is precisely what we require. In the random oracle model (our target setting to prove security), this proof is tight, and essentially comes down to proving that the only valid ciphertexts are the ones created using proper encryption, which requires the use of the random oracle. Because of this, plaintexts can be extracted by looking at the adversary's queries made to the random oracle, and the actual usage of the secret key is not needed. The only possible issue here lies in decryption errors: with this technique of returning plaintexts, a decryption error will never occur, whereas in reality, they occur a negligible fraction of the time. Based on the analysis of [Ava+20], this introduces a minor difference between the advantage of the adversary of $4q_{RO}\delta$, where q_{RO} is the number of queries to the random oracle and δ is the decryption failure rate of Kyber (which for the parameters we have proposed is approximately 2^{-146}). We let ϵ_{dec} denote the distinguishing advantage between the real and simulated decryption oracle. Since $2N$ oracles need simulation, the difference between G_0 and G_1 is $2N \cdot \epsilon_{dec}$.

In game G_2, we replace the signature oracle with one that does not use the secret key, and instead uses the HVZK simulator for the Fiat-Shamir NIZK and reprograms the random oracle accordingly. This is a standard step in proving the EUF-CMA security of Fiat-Shamir based signature schemes, reducing it to the EUF-NMA by simulating the signing oracle without the secret key. Simulation of the signing oracle of Dilithium can be done using the techniques of [KLS18] for signature schemes based on deterministic Fiat-Shamir with abort, which is the case of Dilithium. This simulation of the signing oracle introduces an error[8] of $2^{-\alpha+1} + \kappa_m q_S \cdot \epsilon_{zk}$ which we denote ϵ_{sig}, where α is the min-entropy of w in the Dilithium signature scheme, q_S is the number of signature queries, κ_m is an upper-bound on the number of attempts in the signature algorithm and ϵ_{zk} is the distinguishing advantage between real and simulated transcript for the proof system. Since $2N$ signature oracles are simulated, this introduces a difference of $2N \cdot \epsilon_{sig}$ in the advantage term.

In game G_3, we modify how the keys are expanded. Instead of deriving the expansion secrets through the output of the random oracle as $\tau_i = H(\mathsf{pk}_D^{(b)}, \mathsf{pk}_K^{(b)}, i)$, we simply sample τ_i uniformly at random. This does not change distribution of the secret values at all, and so the only way an adversary can detect such a difference is if it manages to query the input to the random oracle on a point that would otherwise be used to produce the random seed for key expansion. Such a point must necessarily have either $\mathsf{pk}_D^{(b)}$ or $\mathsf{pk}_K^{(b)}$ as a substring for $b \in \{0, 1\}$, and so we can say that games G_2 and G_3 are identical until this event occurs. We will evaluate the probability that this occurs in a moment.

Finally, in game G_4 we replace the S-MLWE samples used for expansion (computed by $\mathsf{Dil.Expand}$ and $\mathsf{Kyb.Expand}$) with uniformly random elements in $\mathcal{R}_q^k$. By the S-MLWE assumption, the difference between game G_3 and G_4 is $\mathsf{Adv}^{\text{S-MLWE}}$. As a result, each public key pair $\rho_1, \rho_2, \ldots, \rho_{2N}$ is just a pair of uniformly random element of $\mathcal{R}_q^k$, with no connection to the original keys $\rho^{(0)}$ or $\rho^{(1)}$. Since at this point nothing the adversary learns depends on the original key pairs, their success probability is limited to that of guessing: $\frac{N-1}{2N-1}$.

All that remains is the difference in the adversary's success probability between games G_2 and G_3, the probability that the adversary queries the random oracle with an input that contains $\mathsf{pk}_D^{(0)}$, $\mathsf{pk}_K^{(0)}$, $\mathsf{pk}_D^{(1)}$ or $\mathsf{pk}_K^{(1)}$. Note that in G_4, the adversary has no information about any of these public keys, so the probability that it queries them to the random oracle is bounded by the min-entropy of these public keys. If we let h_∞ be a lower-bound on this min-entropy, then this probability can be upper-bounded by $4q_{RO}2^{-h_\infty}$. By the Fundamental Lemma of Game-Playing (Lemma 1 of [BR06]), we can thus bound the difference between G_2 and G_3 by $4q_{RO}2^{-h_\infty}$.

By summing the advantages between the games, we get that the overall advantage is

$$4q_{RO}2^{-h_\infty} + 2N(\epsilon_{dec} + \epsilon_{sig}) + \mathsf{Adv}^{\text{S-MLWE}} \quad .$$

$\square$

6.2 Unlinkability Against a Malicious Registration Authority

To prove unlinkability against a malicious RA, we must first show that Kyber retains ciphertext indistinguishability in the presence of expanded keys.

[8]Theorem 3.2 of [KLS18].

6.2.1 Indistinguishability of Expanded Kyber

To model indistinguishability, we modify the normal IND-CCA2 security game for PKEs to introduce features related to key expansion. In particular, the adversary is allowed to make a decryption query with respect to a public key that results from any of the N expansions. The adversary may then request a challenge with respect to any expanded public key of their choosing. It is then provided with the usual PKE IND-CCA2 challenge: the adversary presents two plaintexts and receives an encryption of one of the two. The adversary wins if it is able to guess which it received (possibly after more queries to the decryption oracle).

Our proof below only applies to expansion with respect to a deterministic seed (i.e., the output of a hash on a known input). Therefore, our security game is presented in this setting. It is an interesting open question to prove a reduction from expanded Kyber to that of regular Kyber when the adversary can choose the expansion seed.

Definition 5. The indistinguishability under adaptively chosen ciphertext attack (IND-exCCA2) of a deterministically expanded PKE is defined through the following experiment.

1. The challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.
2. The challenger expands[9] the public key N times:
   - $\mathsf{pk}_i \leftarrow \mathsf{PKE.Expand}(\mathsf{pk}, H(\mathsf{pk}, i))$ for $i \in [N]$
3. Send $(\mathsf{pk}_i)_{i\in[N]}$ to the adversary.
4. The adversary can submit decryption queries to $\mathsf{PKE.Dec}$ for a ciphertext c and for any of the N expanded keys.
5. The adversary requests a challenge for an index i^* and plaintexts (m_0^*, m_1^*).
6. The challenger picks $b \leftarrow_\$ \{0,1\}$ and sends $c^* = \mathsf{PKE.Enc}(\mathsf{pk}_{i^*}, m_b^*)$ to the adversary.
7. The adversary again has oracle access to the N decryption oracles, but is not allowed to query (c^*, i^*).
8. The adversary outputs a guess b' and wins if $b = b'$.

We let $\mathsf{Adv}^{\text{IND-exCCA2}}$ denote the advantage of the adversary in the above game defined as

$$\mathsf{Adv}^{\text{IND-exCCA2}} = \left|\Pr[b = b'] - \frac{1}{2}\right| \ .$$

Theorem 2. *For any* PPT *adversary,*

$$\mathsf{Adv}_{\mathsf{Kyb}}^{\text{IND-exCCA2}} \leq N \cdot \mathsf{Adv}_{\mathsf{Kyb}}^{\text{IND-CCA2}} + \mathsf{Adv}_{B_{\eta_1}}^{\text{MLWE}} + 4q_{RO}\delta \ .$$

To prove security, we will show that an adversary's advantage in breaking the IND-CCA2 security *with* the expanded keys translates to an advantage in breaking the IND-CCA2 security *without* the key expansion, i.e. against the PKE obtained from Kyber's KEM and an IND-CCA2 symmetric key encryption. We do this by guessing the index i^* that the adversary will choose for their challenge and by reprogramming the random oracle such that if our guess is right, the challenge comes directly from a "plain" Kyber instance, and thus the adversary's distinguishing ability directly translates to their ability to break Kyber. Since we must correctly guess the index i^* for which the challenge is requested (out of N), this results in a tightness loss of N, the number of times that the caterpillar key pk is expanded. We leave it as an open question to find a proof technique that closes this tightness gap.

[9]Although the RA that expands the Kyber public key, this expansion is with respect to a deterministic seed $\tau = H(\mathsf{pk}, i)$, so it has no control over the expanded keys. We omit the signature public key from the oracle input that would be present in the BKE protocol.

Proof of Theorem 2. Given an adversary $\mathcal{A}$ against the indistinguishability of expanded Kyber, we construct a reduction $\mathcal{R}$ against the indistinguishability of "regular" Kyber. The reduction plays two games simultaneously; it plays the role of the challenger in the IND-exCCA game with $\mathcal{A}$ and plays the role of the adversary in the IND-CCA2 game with a challenger $\mathcal{C}$. The reduction is provided by $\mathcal{C}$ with a Kyber public key $\hat{\mathsf{pk}}_K$, which is an MLWE sample with distribution B_{η_1}. Before giving anything to the adversary $\mathcal{A}$, the reduction selects an index $\hat{\imath} \leftarrow_\$ \{1, \ldots, N\}$. $\mathcal{R}$ then selects a uniform 256 bits $\hat{\tau} \leftarrow_\$ \{0,1\}^{256}$ and samples s' and e' as in our Expand functionality (see Figure 2) with $\hat{\tau}$ as the seed. This allows $\mathcal{R}$ to compute t' (in the NTT domain or not) and to set the identity public key pk in the IND-exCCA game as $\hat{\mathsf{pk}}_K - t'$. Further, $\mathcal{R}$ reprograms the random oracle so that $H(\mathsf{pk}, \hat{\imath}) = \hat{\tau}$, such that the $\hat{\imath}$th expanded key will equal $\hat{\mathsf{pk}}_K$. Since this is done prior to starting the interaction with the adversary (and the distribution of $\hat{\tau}$ is uniform), there is no ability for the adversary to notice this reprogramming. This guarantees that the expanded public key is equal to $\hat{\mathsf{pk}}_K$ with probability $\frac{1}{N}$.

We must establish that the adversary cannot notice that the original public key pk is generated in this way. First we consider the distribution of the public key itself. Intuitively, the adversary expects an MLWE sample, and receives the difference of two (independent) MLWE samples. So long as the adversary is unable to distinguish MLWE samples from uniform (which is our security assumption), it should be unable to tell the difference between the two. In particular, it is the challenge public key $\hat{\mathsf{pk}}_K$ which we need to be indistinguishable from uniform. This means that $\hat{\mathsf{pk}}_K - t'$ is also indistinguishable from uniform. This introduces a $\mathsf{Adv}^{\mathrm{MLWE}}_{B_{\eta_1}}$ term in the reduction's advantage.

Next we need to work out how $\mathcal{R}$ responds to decryption and random oracle queries. The general idea is that the random oracle will give us a way to "break up" any queries into those relevant to the challenge public key (queries made with respect to index $\hat{\imath}$) and all other queries. Relevant random oracle queries can be forwarded to the challenger's random oracle, to maintain consistency between the two games $\mathcal{R}$ is playing.

Decryption queries must be made with respect to an index $i \in [N]$. For queries corresponding to the target public key $\hat{\mathsf{pk}}_K$ (i.e., for index $\hat{\imath}$), we can forward the decryption queries to the external IND-CCA2 decryption oracle for $\hat{\mathsf{pk}}_K$. For any index other than $\hat{\imath}$, we can use the knowledge extractor that exists for Fujisaki-Okamoto based PKEs to answer the queries without knowing the secret key. The reduction looks at the random oracle queries the adversary made to figure out which message corresponds to the query ciphertext, as we did in the CA unlinkability proof. As in that proof, this introduces a $4q_{RO}\delta$ term which corresponds to the adversary's ability to construct an ciphertext that should not decrypt.

Finally, when a challenge is requested for index i^*, we hope that the request is issued with respect to $\hat{\imath}$, which happens with probability $1/N$. Assuming this happens, we can forward the query (m_0^*, m_1^*) to the external IND-CCA2 challenger, and receive back a challenge ciphertext c^* that is valid with respect to $\hat{\mathsf{pk}}_K$. The adversary's ability to win their game then directly translates into the reduction's ability to win the game with the external challenger. Summing up, we have

$$\mathsf{Adv}^{\mathrm{IND\text{-}exCCA2}}_{\mathsf{Kyb}} \leq N \cdot \mathsf{Adv}^{\mathrm{IND\text{-}CCA2}}_{\mathsf{Kyb}} + \mathsf{Adv}^{\mathrm{MLWE}}_{B_{\eta_1}} + 4q_{RO}\delta \ .$$

□

6.2.2 RA Unlinkability

We are now ready to prove that the BKE protocol instantiated with our expanded Kyber and Dilithium schemes is unlinkable against malicious RA (Definition 3).

Theorem 3. *The butterfly key expansion scheme of Figure 1 is unlinkable against malicious*

RA (Definition 3) with advantage

$$\mathsf{Adv}^{\text{UL-RA}} \le \mathsf{Adv}^{\text{IND-exCCA2}}_{\mathsf{Kyb}} + \mathsf{Adv}^{\text{UF-CMA}}_{\mathsf{Dil}} + \mathsf{Adv}^{\text{MLWE}}_{S_\eta}$$

Proof. We proceed with a game hopping argument. Game G_0 corresponds to the unlinkability game from Definition 3.

In game G_1, instead of doing the regular checks in step 6 of the game, the challenger aborts if $pkg'_i \neq pkg_i$ for some $i \in [N]$. Since the challenger checks that sig'_i is a valid signature of pkg'_i for the CA's public key in step 6, the outcomes of games G_0 and G_1 differ only if the RA can forge a signature sig'_i of $pkg'_i \neq pkg_i$ for some i. By the unforgeability of Dilithium, this can happen with probability at most $\mathsf{Adv}^{\text{UF-CMA}}_{\mathsf{Dil}}$.

In game G_2, we replace how the ciphertexts pkg_i are generated. Instead of encrypting (σ_i, τ'_i) as the CA would (see Figure 1), the challenger encrypts the all 0 string. In step 4 of the game, the challenger sets $pkg_i = \mathsf{Kyb.Enc}(\mathsf{pk}_{K,i}, 0^\ell)$ for $i \in [N]$, where $\ell = |\sigma_i| + |\tau'_i|$ is the bit length of the encrypted message. By the ciphertext indistinguishability of expanded Kyber (Definition 5), the malicious RA should not be able to detect this change in how pkg_i is constructed. During the unpacking of the certificates (step 6), the challenger still uses the true values for τ'_i and σ_i. The indistinguishability of G_1 and G_2 follows from the IND-exCCA2 security of expanded Kyber (Theorem 2):

$$|\Pr[b = b' \mid G_0] - \Pr[b = b' \mid G_1]| \le \mathsf{Adv}^{\text{IND-exCCA2}}_{\mathsf{Kyb}}$$

In game G_3, instead of computing $\mathsf{pk}'_{D,i}$ as $\mathsf{Dil.Expand}(\mathsf{pk}_{D,i}, \tau'_i)$ when constructing pkg_i, which adds a random MLWE instance to $\mathsf{pk}_{D,i}$, the challenger adds a uniformly random value. The indistinguishability of games G_2 and G_3 follows from the S-MLWE assumption with distribution S_η.

$$|\Pr[b = b' \mid G_3] - \Pr[b = b' \mid G_2]| \le \mathsf{Adv}^{\text{S-MLWE}}_{S_\eta} \quad .$$

What the adversary gets in game G_3 is now unrelated to the original public key pk_D: the expanded keys $\mathsf{pk}'_{D,i}$ are uniformly random, so the probability that the adversary can distinguish $b = 0$ from $b = 1$ is exactly $\frac{1}{2}$.

Summing up, we have that the advantage of the adversary in the unlinkability game is at most

$$\mathsf{Adv}^{\text{IND-exCCA2}}_{\mathsf{Kyb}} + \mathsf{Adv}^{\text{UF-CMA}}_{\mathsf{Dil}} + \mathsf{Adv}^{\text{MLWE}}_{S_\eta}$$

□

6.3 Unforgeability of Expanded Dilithium

For unforgeability, we only require a single proof of security since we want to ensure that even when the RA and CA collude, they cannot forge a valid signature for an expanded public key accepted by the vehicle. Thus, the adversary is permitted to provide expansion packages to the vehicle and then request signatures with respect to the resulting public keys. The adversary wins if they are able to forge a signature with respect to any expanded key.

Theorem 4. *Suppose* $\mathsf{ExpandS}$ *is a random oracles. Let Q be an upper-bound on the number of random oracle queries and let N be the number of expanded public keys. Then the butterfly key expansion scheme instantiated with expanded Dilithium is unforgeable under chosen key expansion and signature attack (Definition 4) with advantage*

$$\mathsf{Adv}^{\text{UF-exCMA}} \le Q^2 \cdot \mathsf{Adv}^{\text{UF-CMA}}_{\mathsf{Dil}} + (N+1) \cdot \mathsf{Adv}^{\text{MLWE}}_{S_\eta} + N \cdot \epsilon_{sig}$$

Proof. There are two rounds of expansion with Dilithium, as opposed to the single round with Kyber. Nonetheless, we can proceed similarly to how we proved the indistinguishability of Kyber for Theorem 2: by injecting a challenge public key into the adversary's random oracle queries.

Consider an attacker $\mathcal{A}$ against the unforgeability of expanded Dilithium (Definition 4). We construct a reduction $\mathcal{R}$ that uses $\mathcal{A}$ to produce a forgery against the regular chosen message unforgeability of Dilithium. The reduction is provided with a Dilithium public key $\hat{\mathsf{pk}}_D$ by an external challenger and plays the dual role of the adversary in the UF-CMA game for Dilithium and of the challenger in the UF-exCMA game (Definition 4) played with $\mathcal{A}$. This means it has access to a signature oracle for $\hat{\mathsf{pk}}_D$ and to an external random oracle $\mathcal{O}$ for the UF-CMA game. The reduction controls the random oracles $\mathsf{ExpandS}$ and H used by $\mathcal{A}$.

Before interacting with $\mathcal{A}$ or providing it with any oracle access, the reduction does the following oracle reprogramming. The reduction first samples (s_1, s_2) and (s_1', s_2') from the distribution $S_\eta^\ell \times S_\eta^k$ (recall that S_η is the uniform distribution over $[-\eta, \eta]$), and it computes the offsets $t \leftarrow \hat{\mathsf{pk}}_D - As_1 - s_2$ and $t' \leftarrow t - As_1' - s_2'$ and the key $\mathsf{pk} \leftarrow \hat{\mathsf{pk}}_D - t - t'$. Let Q be an upper-bound on the number of queries to $\mathsf{ExpandS}$ by $\mathcal{A}$ and recall that N is the number of expanded keys. The reduction selects two random indices $\hat{\imath}, \hat{\jmath} \leftarrow_\$ [Q]$ such that $\hat{\imath} < \hat{\jmath}$. When the adversary makes their $\hat{\imath}$th (resp. $\hat{\jmath}$th) query to $\mathsf{ExpandS}$ with an input $\tau_{\hat{\jmath}}$ (resp. $\tau_{\hat{\jmath}}$), we set the output to be (s_1, s_2) (resp. (s_1', s_2')). The reduction's gamble is that the adversary chooses to make their forgery using seeds $\tau_{\hat{\imath}}$ and $\tau_{\hat{\jmath}}$ corresponding to the $\hat{\imath}$th and $\hat{\jmath}$th query, respectively. If this is the case, then the forgery is for the target Dilithium public key $\mathsf{pk} + t + t' = \hat{\mathsf{pk}}_D$.

Now, the reduction proceeds as the challenger in the game of Definition 4: in the first step, it sends $\mathsf{pk} = \hat{\mathsf{pk}}_D - t - t'$ to $\mathcal{A}$. Note that the distribution of public key pk is not the same as in the orginial game, however this change is undetectable to the adversary from the same logic as in the case of Kyber (see the proof of Theorem 2); by the MLWE assumption, both the real distribution of pk and the distribution of pk computed by $\mathcal{R}$ are indistinguishable from uniformly random. This introduces a $\mathsf{Adv}_{S_\eta}^{\mathrm{MLWE}}$ additive loss in $\mathcal{R}$'s success probability.

In step 2 of the game, for each seed τ sent by the adversary, $\mathcal{R}$ rejects if τ was not queried to $\mathsf{ExpandS}$. The probability that $\mathcal{R}$ rejects a seed for which would have been accepted by the challenger in the UF-exCMA game is at most $N \cdot \mathsf{Adv}_{S_\eta}^{\mathrm{MLWE}}$ by the following argument.

Since the challenger computes pk_i' as $\mathsf{pk}_i' = \mathsf{Expand}(\mathsf{Expand}(\mathsf{pk}, \tau_i), \tau_i')$. If either of τ_i or τ_i' is not queried to $\mathsf{ExpandS}$ by $\mathcal{A}$, then the values for (s_1, s_2) and (s_1', s_2') computed using the random oracle $\mathsf{ExpandS}$ are unknown to the adversary. Because Expand adds $A \cdot s_1 + s_2$ and $A \cdot s_1' + s_2'$ to the public key, by the MLWE assumption the expanded public key is indistinguishable from random to the adversary. The probability that the adversary can guess the value of the doubly expanded public key pk_i' without knowing the expansion factors (s_1, s_2) and (s_1, s_2) is at most $\mathsf{Adv}_{S_\eta}^{\mathrm{MLWE}}$. By a union bound over the N packages, the probability that there is a package rejected by $\mathcal{R}$ that would be accepted by the challenger in the game is at most $N \cdot \mathsf{Adv}_{S_\eta}^{\mathrm{MLWE}}$.

Still in step 2 of the game, the reduction aborts if there is no index i such that $\mathsf{pk}_i' = \hat{\mathsf{pk}}_D$. This happens if the reduction is wrong in its guess for $\hat{\imath}$ and $\hat{\jmath}$ which happens with probability $1 - \frac{1}{Q^2}$.

We now assume that the reduction does not abort in step 2 and describe the rest of its strategy. In step 3, whenever the adversary asks for a signature of m for an index i, the reduction does the following. If the expanded key pk_i' is equal to $\hat{\mathsf{pk}}_D$, i.e. the challenge Dilithium public key for which the reduction tries to build a forgery, then the reduction submits m to the external signature oracle for $\hat{\mathsf{pk}}_D$ of the UF-CMA game it is playing. For any other index, the reduction must simulate the signature oracle for Dilithium without access to the secret key. This is done by using the Fiat-Shamir HVZK simulator to produce a valid Fiat-Shamir transcript and by reprogramming the random oracle H used in the signature algorithm. The simulation of a signature oracle introduces an error ϵ_{sig} (see proof of Theorem 1 for a description of this error term). Since $\mathcal{R}$ simulates N oracles, the loss in success probability is $N \cdot \epsilon_{sig}$.

A remaining challenge is to argue that the RO reprogramming done by $\mathcal{R}$ to simulate the signature oracles for pk'_i does not affect the probability that the forgery for $\hat{\mathsf{pk}}_D$ is valid with respect to the external (unprogrammed) random oracle from the UF-CMA game. The adversary has access to the RO to make its forgery, and so it should have access to the external random oracle $\mathcal{O}$ to make a forgery for the target public key $\hat{\mathsf{pk}}_D$. There are 2 types of RO queries that occur in the signing and verification algorithms of Dilithium[10]: the first query computes $\mu \leftarrow H(H(\mathsf{pk})\|M)$ and then every oracle query contains μ.

To make sure only queries related to the target public key $\hat{\mathsf{pk}}_D$ are forwarded to the external RO, $\mathcal{R}$ first computes $\hat{h} = \mathcal{O}(\hat{\mathsf{pk}}_D)$. It then offers an RO interface H to the adversary in the following way:

- for any query x prefixed by $\hat{h}$, forward x to $\mathcal{O}$ and record the result μ;
- for any query x that contains a recorded μ as a substring, forward x to $\mathcal{O}$;
- for every other query x, simulate the RO using lazy sampling (i.e. sample $H(x)$ uniformly at random if it is not already defined).

Thus the reduction $\mathcal{R}$ can both respond to signature queries made by $\mathcal{A}$, and the forgery that the adversary submits will correspond to a forgery for the challenge public key $\hat{\mathsf{pk}}_D$, so long as we guessed the indices $\hat{i}$ and $\hat{j}$ correctly.

When the adversary submits its forgery (m^*, σ^*, i^*), the reduction outputs (m^*, σ^*) if i^* corresponds to the key expanded from $\tau_{\hat{i}}$ and $\tau_{\hat{j}}$ (if $\mathsf{pk}'_{i^*} = \hat{\mathsf{pk}}_D$) and otherwise aborts. If the reduction made it to this point, the probability it does not abort is at least $\frac{1}{Q^2}$.

Concluding, we have that if $\mathcal{A}$ produces a forgery, then $\mathcal{R}$ produces a forgery with probability at least $\frac{1}{Q^2}$. By accounting for the times we have invoked the MLWE assumption, it follows that

$$\mathsf{Adv}^{\text{UF-exCMA}} \leq Q^2 \cdot \mathsf{Adv}_{Dil}^{\text{UF-CMA}} + (N+1) \cdot \mathsf{Adv}_{S_\eta}^{\text{MLWE}} + N \cdot \epsilon_{sig}$$

□

7 Conclusion

We have established a formal framework for proving the security of the butterfly key expansion pseudonymous certificate provisioning process. Our expanded Kyber and Dilithium algorithms introduced in this context provide a secure instantiation of BKE. They may also find applications in other areas employing public key rerandomization for privacy.

Some of our security reductions incur a tightness loss of N, the number of issued certificates. The question arises whether this loss is inherent or just a consequence of our proof techniques. Our proofs hold in the random oracle model, prompting the question of the applicability of recent techniques in the *quantum* random oracle model – where the adversary has quantum access to the oracle – to our proofs for full post-quantum security.

[10]We refer to the Dilithium specification [Duc+21] for the description of those algorithms.

8 Acknowledgements

This research was part of the "Quantum Secure Cryptographic Primitives with Applications to Vehicle-To-Vehicle Networks" project funded by the NRC-Waterloo Collaboration Centre Funding Program.

References

[Alk+20] Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Juliane Krämer, Patrick Longa, and Jefferson E. Ricardini. "The Lattice-Based Digital Signature Scheme qTESLA". In: *Applied Cryptography and Network Security*. Ed. by Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi. Cham: Springer International Publishing, 2020, pp. 441–460. ISBN: 978-3-030-57808-4. DOI: 10.1007/978-3-030-57808-4_22.

[Ava+20] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. *CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation*. Version 3.0. 2020. URL: https://pq-crystals.org/kyber/resources.shtml (visited on 08/21/2023).

[Bar+18] Paulo S. L. M. Barreto, Jefferson E. Ricardini, Marcos A. Simplicio Jr., and Harsh Kupwade Patil. "qSCMS: Post-quantum certificate provisioning process for V2X". In: *IACR Cryptol. ePrint Arch.* (2018), p. 1247. URL: https://eprint.iacr.org/2018/1247.

[BR06] Mihir Bellare and Phillip Rogaway. "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs". In: *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings*. Vol. 4004. Lecture Notes in Computer Science. Springer, 2006, pp. 409–426. DOI: 10.1007/11761679_25.

[BR95] Mihir Bellare and Phillip Rogaway. "Optimal asymmetric encryption". In: *Advances in Cryptology — EUROCRYPT'94*. Ed. by Alfredo De Santis. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1995, pp. 92–111. ISBN: 978-3-540-44717-7. DOI: 10.1007/BFb0053428.

[Can19] Transport Canada. *Security Credential Management System (SCMS) – Requirements Analysis Report*. Nov. 1, 2019. URL: https://tc.canada.ca/en/innovation-centre/priority-reports/security-credential-management-system-scms-requirements-analysis-report (visited on 02/27/2024).

[Che+20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. *NTRU*. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. 2020.

[CS03] Ronald Cramer and Victor Shoup. "Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack". In: *SIAM Journal on Computing* 33.1 (Jan. 2003), pp. 167–226. ISSN: 0097-5397. DOI: 10.1137/S0097539702403773. (Visited on 02/05/2024).

[DS21] Leo Ducas and John Schanck. *Security Estimation Scripts for Kyber and Dilithium*. available at https://github.com/pq-crystals/security-estimates. Accessed September 2023. 2021.

[Duc+21] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. *CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation.* Version 3.1. 2021. URL: https://pq-crystals.org/dilithium/resources.shtml (visited on 08/21/2023).

[ESS21] Edward Eaton, Douglas Stebila, and Roy Stracovsky. "Post-Quantum Key-Blinding for Authentication in Anonymity Networks". In: *Progress in Cryptology – LATINCRYPT 2021.* Ed. by Patrick Longa and Carla Ràfols. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 67–87. ISBN: 978-3-030-88238-9. DOI: 10.1007/978-3-030-88238-9_4.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Advances in Cryptology — CRYPTO' 99.* Ed. by Michael Wiener. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pp. 537–554. ISBN: 978-3-540-48405-9. DOI: 10.1007/3-540-48405-1_34.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. "Public-key cryptosystems from lattice reduction problems". In: *Advances in Cryptology — CRYPTO '97.* Ed. by Burton S. Kaliski. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1997, pp. 112–131. ISBN: 978-3-540-69528-8. DOI: 10.1007/BFb0052231.

[HP01] Stuart Haber and Benny Pinkas. "Securely combining public-key cryptosystems". In: *Proceedings of the 8th ACM conference on Computer and Communications Security.* CCS '01. New York, NY, USA: Association for Computing Machinery, Nov. 5, 2001, pp. 215–224. ISBN: 978-1-58113-385-1. DOI: 10.1145/501983.502013.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. "A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model". In: *Advances in Cryptology – EUROCRYPT 2018.* Ed. by Jesper Buus Nielsen and Vincent Rijmen. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pp. 552–586. ISBN: 978-3-319-78372-7. DOI: 10.1007/978-3-319-78372-7_18.

[LLC16] Crash Avoidance Metrics Partners LLC. *Security Credential Management System Proof-of-Concept Implementation—EE Requirements and Specifications Supporting SCMS Software Release 1.1.* 2016. URL: https://www.campllc.org/security-credential-management-system-scms-proof-of-concept-poc-implementation/ (visited on 02/27/2024).

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. "On Ideal Lattices and Learning with Errors over Rings". In: *J. ACM* 60.6 (2013), 43:1–43:35. DOI: 10.1007/978-3-642-1.

[PW11] Chris Peikert and Brent Waters. "Lossy Trapdoor Functions and Their Applications". In: *SIAM Journal on Computing* 40.6 (Jan. 2011), pp. 1803–1844. ISSN: 0097-5397. DOI: 10.1137/080733954.

[SB23] Ahmad Salman and Zachary Blankinship. "Implementing Butterfly Key Expansion Using Post-Quantum Algorithms". In: *Proceedings of Seventh International Congress on Information and Communication Technology.* Ed. by Xin-She Yang, Simon Sherratt, Nilanjan Dey, and Amit Joshi. Lecture Notes in Networks and Systems. Singapore: Springer Nature, 2023, pp. 507–516. ISBN: 978-981-19160-7-6. DOI: 10.1007/978-981-19-1607-6_45.

[Sim+18] Marcos A. Simplicio, Eduardo Lopes Cominetti, Harsh Kupwade Patil, Jefferson E. Ricardini, and Marcos Vinicius M. Silva. “The Unified Butterfly Effect: Efficient Security Credential Management System for Vehicular Communications”. In: *2018 IEEE Vehicular Networking Conference (VNC)*. 2018 IEEE Vehicular Networking Conference (VNC). ISSN: 2157-9865. Dec. 2018, pp. 1–8. DOI: 10.1109/VNC.2018.8628369.

[SM17] Douglas Stebila and Michele Mosca. “Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project”. In: *Selected Areas in Cryptography – SAC 2016*. Ed. by Roberto Avanzi and Howard Heys. Cham: Springer International Publishing, 2017, pp. 14–37. ISBN: 978-3-319-69453-5. DOI: 10.1007/978-3-319-69453-5_2.

[Uni19] United States. Department of Transportation. Intelligent Transportation Systems Joint Program Office. *Connected Vehicle Deployment Technical Assistance: Security Credential Management System (SCMS) Technical Primer*. FHWA-JPO-19-775. Nov. 1, 2019. URL: https://rosap.ntl.bts.gov/view/dot/43635 (visited on 02/27/2024).

[Why+13] William Whyte, André Weimerskirch, Virendra Kumar, and Thorsten Hehn. “A security credential management system for V2V communications”. In: *2013 IEEE Vehicular Networking Conference*. 2013 IEEE Vehicular Networking Conference. ISSN: 2157-9865. Dec. 2013, pp. 1–8. DOI: 10.1109/VNC.2013.6737583.

A Key Reuse/Unified BKE

A.1 Unsafe Reuse of Key Pairs

For textbook RSA-based schemes, the encryption algorithm is defined as $\mathsf{Enc}(m) = m^e \mod N$ and $\mathsf{Dec}(c) = c^d \mod N$; and the signature algorithm as $\mathsf{Sign}(m) = m^d \mod N$ and $\mathsf{Ver}(\sigma, m) = m \overset{?}{=} [\sigma^e \mod N]$. As we can see, a query to the decryption oracle on input m provides a valid signature for m.

The same is true for some lattice-based signature and encryption schemes, in particular those of [GGH97]. The encryption scheme takes a point on the lattice determined by the message and adds noise to it. The decryption algorithm maps a ciphertext in $\mathbb{R}^n$ to its closest point on the lattice. The signing algorithm takes a point from $\mathbb{R}^n$ and maps it to the closest point on the lattice. Verification is done by checking that the distance is at most some threshold. We can see again that the decryption oracle provides a valid signature.

A.2 Safely Reusing Keys for Unified Butterfly Key Expansion

We rely on the definitions of [HP01]. A *combined signature and encryption scheme* is a tuple $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver}, \mathsf{Enc}, \mathsf{Dec})$ such that $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Ver})$ forms a public-key signature scheme and $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ a public-key encryption scheme (or KEM). We ask that the encryption and signature schemes individually satisfy stronger notions of security than the typical notions to ensure that we can safely use the same key pair in both settings. We say that the signature scheme that it is *UF-CMA in the presence of a decryption oracle* if in the UF-CMA game, the adversary also has oracle access to $\mathsf{Dec}(\mathsf{sk}, \cdot)$ in addition to the signature oracle and still cannot forge a signature. Similarly, in the IND-CCA game, the adversary has oracle access to $\mathsf{Sign}(\mathsf{sk}, \cdot)$. We say that it is *IND-CCA in the presence of a signature oracle* if it still cannot distinguish ciphertexts in this setting.

Below, we describe a general framework for safely using the same key pair for both digital signatures and public-key encryption. Since Dilithium and Kyber fit this framework, it is in principle possible to use the same key pair for both and thus obtain a *unified* version of butterfly key expansion. We leave to future work the task of whether the parameters of both schemes can be tweaked such that they remain secure and the size of the combined public key is smaller than the sum of Kyber's and Dilithium's public key sizes (yielding a reduction in bandwidth). In particular, the modulus of both schemes differ by a large amount $q = 3329$ for Kyber and $q = 8380417$ for Dilithium.

A.2.1 Security of Signature Schemes in the Presence of a Fujisaki-Okamoto Decryption Oracle

Haber and Pinkas [HP01] have shown that in the random oracle model, signature scheme retain their security when combined[11] with encryption schemes based on the optimal asymmetric encryption padding of Bellare and Rogaway [BR95]. The proof relies on the *plaintext awareness* property of [BR95], which essentially says that for any valid ciphertext produced by the adversary, the adversary should "know" the corresponding plaintext. This is formalized using the concept of knowledge extraction, a PPT algorithm $\mathcal{K}$ knowing the ciphertext and the oracle queries made by the adversay that can decrypt the ciphertext with good probability. The Fujisaki-Okamoto transform [FO99] on which the CCA secure variant of Kyber is based admits such a knowledge extractor. Concretely this means that decryption queries can be answered without the secret key by recording and reprogramming random oracle queries made by the adversary, so the unforgeability of Dilithium in the presence of a Kyber decryption oracle can be reduced to the UF-CMA security of Dilithium.

A.2.2 Security of Encryption Schemes in the Presence of a Fiat-Shamir Signing Oracle

For signature schemes based on the Fiat-Shamir transform, if the underlying identification scheme is HVZK, then UF-CMA and UF-NMA are tightly equivalent in the ROM since the signature oracle can be simulated using the HVZK simulator together with oracle reprogramming. By the same argument as above, the queries to the signature oracle in the game of ciphertext indistinguishability in the presence of signature oracle can be answered without the secret key in the ROM, so this notion tightly reduces to the usual IND-CCA notion.

B Additional Proofs

Lemma 1. *Assume the* MLWE *problem is hard, then so is the* S-MLWE *problem.*

Proof. Let $\mathcal{A}$ be an adversary against the S-MLWE problem with N samples. Let $A \leftarrow\$ R_q^{m\times k}$ be a random matrix of polynomials. We construct a series of N hybrids $H_0, \dots, H_N$ as follows. Let u_i, s_i, e_i be as in Definition 1,

$$H_i = \begin{cases} (A, As_1 + e_1, \dots, As_N + e_N) & i = 1 \\ (A, u_1, \dots, u_{i-1}, As_i + e_i, \dots, As_N + e_N) & i = 2, \dots, N-1 \\ (A, u_1, \dots, u_N) & i = N \end{cases} \tag{1}$$

For all $i \in [N]$, we have that H_i and H_{i+1} are indistinguishable in polynomial time by the MLWE assumption: they differ only in the ith entry which is $As_i + e_i$ in the first case and

[11] By "combined", we mean that the public/private key pairs of both schemes are correlated arbitrarily.

u_i in the second. Therefore

$$\begin{aligned}
&|\Pr[A(H_1) = 1] - \Pr[A(H_N) = 1]| && (2)\\
&\leq \sum_i |\Pr[A(H_i) = 1] - \Pr[A(H_{i+1}) = 1]| && (3)\\
&\leq N \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathrm{MLWE}} && (4)
\end{aligned}$$

□