cryptography

MDPI

Article

DPS: A Post-Quantum Proxy Signature Scheme from Dilithium for IoT Applications

Yuteng Wang [1], Ruoyu Ding [1], Tianrun Yu [1], Zhen Han [1], Jian Weng [2,*] and Jiasi Weng [2]

[1] College of Cyber Security, Jinan University, Guangzhou 510630, China; walwork@stu.jnu.edu.cn (Y.W.); ryding@stu2023.jnu.edu.cn (R.D.); yuterry@stu2024.jnu.edu.cn (T.Y.); hanzhen@stu.jnu.edu.cn (Z.H.)
[2] School of Computer Science and Cyber Engineering, Guangzhou University, Guangzhou 510006, China; wengjiasi@gmail.com
* Correspondence: cryptjweng@gmail.com

Abstract

Proxy signatures enable the secure delegation of signing authority, which is particularly useful in resource-constrained Internet of Things (IoT) environments. However, most existing schemes rely on classical hardness assumptions and therefore cannot resist quantum attacks. To address the challenge, we propose a post-quantum proxy signature scheme based on Dilithium for IoT scenarios. We first propose an asynchronous remote key generation (ARKG) scheme based on CRYSTALS-Kyber, enabling the delegator and proxy signer to generate proxy keys of Dilithium without real-time interaction. We further integrate ARKG with the Dilithium signature scheme to construct a proxy signature scheme called DPS while ensuring the unlinkability of proxy signatures. Additionally, our proposed DPS achieves post-quantum security and provides unforgeability, distinguishability, verifiability, and undeniability with formal proofs. Experimental performance evaluation shows that our scheme yields significant efficiency gains over existing quantum-safe proxy signature solutions, with 10× speedup for both the delegation and proxy signing phases, as well as a 2.4× improvement in the verification phase.

Keywords: proxy signature; post-quantum cryptography; Dilithium-QROM; lattice-based cryptography; CRYSTALS-Kyber; Internet of Things

1. Introduction

Internet of Things (IoT) technology connects billions of intelligent devices across domains such as smart homes, the industrial IoT, and smart healthcare, generating and transmitting vast amounts of data [1,2]. In IoT environments, it is common that devices cannot be legitimately accessed by third parties when the device owner is unavailable or that devices themselves lack sufficient computational resources to handle a large volume of requests. Due to limited computational capability or operational availability, a more powerful proxy—such as a fog node or edge server—could be delegated to cope with these situations, thereby improving operational flexibility, alleviating computational burden and reducing system latency.

Proxy signature schemes in cryptography provide a natural primitive for enabling such delegated authorization. A proxy signature scheme, firstly introduced by Mambo et al. [3], is a specialized digital signature primitive that allows an original signer to delegate their signing authority to a proxy agent. It enables a proxy to generate valid signatures on behalf of the original signer, which can be publicly verified using the original signer's public key.

Academic Editors: Xianhui Lu and Jianfeng Wang

Received: 17 March 2026
Revised: 30 April 2026
Accepted: 11 May 2026
Published: 15 May 2026

Early proxy signature schemes were essentially full delegation signatures, in which the original signer directly provides its private key to the proxy signer [4]. Such a design is clearly insecure in practical delegation scenarios, particularly in IoT environments. A more practical approach is delegation by warrant, where the original signer does not reveal its private key but instead issues an unforgeable delegation credential containing the description of the delegated rights and a signature on the proxy's public key [4]. However, this kind of schemes mostly rely on the proxy signer's long-term identity and typically require synchronous interaction, which is unrealistic for resource-constrained IoT devices. To address this limitation, Frymann et al. proposed the cryptographic primitive asynchronous remote key generation (ARKG), which enables non-interactive key derivation [5,6]. They also constructed a certificate-based proxy signature scheme in which the delegation credential signs derived and unlinkable public keys rather than a fixed identity, by integrating ARKG into the Proxy Signature with Unlinkable Warrant (PSUW) framework [7].

Existing proxy signature schemes can also be broadly categorized by their underlying hardness assumptions. Some schemes rely on classical mathematics problems such as discrete logarithms (DLP), integer factorization, or elliptic curve variants (ECC) [3,8,9]. Other classical constructions employ Bilinear Pairings for application like the SE-IDPSC-CS scheme [10] used in cloud data sharing. However, all these schemes are vulnerable to quantum attacks.

To address this threat, post-quantum cryptography (PQC), which aims to be quantum-safe, has been developed. In 2022, the National Institute of Standards and Technology (NIST) selected lattice-based CRYSTALS-Kyber (Kyber) [11] and CRYSTALS-Dilithium (Dilithium) [12] for PQC standardization, which were later standardized as the Module-Lattice-based Key-Encapsulation Mechanism (ML-KEM) [13] and Module-Lattice-based Digital Signature Algorithm (ML-DSA) [14] in 2024, respectively. PQC schemes based on MLWE achieve a good balance between efficiency and storage, making them widely suitable for resource-constrained IoT scenarios [15]. For strong security guarantees, CRYSTALS-Dilithium was extended to a variant called Dilithium-QROM with tight security reductions in the Quantum Random Oracle Model (QROM) [16]. At the same time, various identity-based proxy signature schemes based on lattice have been proposed [17–19]. These schemes use identity strings (e.g., email addresses) as public keys with the risk of potential illegal impersonation or unauthorized signing [20] and usually necessitate secure channels for secret key distribution. Moreover, they suffer the significant computational overhead of the lattice trapdoor. Beyond this, existing ARKG instantiations remain largely grounded in classical hardness assumptions [6], and the extension to lattice-based cryptography is limited to asynchronous key generation of sKEM (split KEM) [5], leaving lattice-based signature schemes without a dedicated instantiation. Motivated by the trapdoor-free structure of Dilithium-QROM and the PSUW framework, we propose DPS, an efficient post-quantum proxy signature scheme, addressing the synchronization challenges and privacy risks in IoT scenarios. By reconstructing ARKG to enable asynchronous remote key generation for Dilithium-QROM, DPS preserves the asynchrony and unlinkability properties of PSUW while completely eliminating the costly trapdoor computations, thereby significantly reducing the overhead for resource-constrained IoT devices.

1.1. Contribution

Motivated by the need to integrate lattice-based cryptographic primitives into practical IoT applications, we propose a new proxy signature scheme based on Dilithium-QROM tailored for IoT environments, with improved implementation efficiency over other lattice-based works. Our main contributions are as follows:

- We proposed an ARKG scheme based on Kyber for the asynchronous key generation of Dilithium-QROM. It utilizes Kyber as its underlying module to derive, in a one-way manner, a key pair suitable for Dilithium-QROM. The derived key pair can be used as the input to the Dilithium-QROM signing algorithm without compromising its correctness or security guarantees.
- We give the first Dilithium-based proxy signature scheme, denoted as DPS, by instantiating the PSUW framework with our proposed ARKG scheme and Dilithium-QROM. The proposed DPS achieves post-quantum security and provides verifiability, unforgeability, undeniability, and unlinkability, all supported by formal security proofs. Furthermore, constructing proxy signatures directly from Dilithium eliminates the need for lattice trapdoors, leading to improved efficiency.
- We provide an experimental evaluation of the proposed scheme using concrete parameter sets. The results demonstrate that our construction has a greater advantage over other lattice-based schemes in terms of proxy signature efficiency. Specifically, experimental results show that our scheme achieves approximately a 10× speedup for both the delegation and proxy signing phases, as well as a 2.4× performance improvement in the verification phase over comparable works.

1.2. Related Work

Proxy signatures, as a cryptographic primitive enabling the secure delegation of signing rights, hold significant application value in distributed scenarios such as the Industrial Internet of Things (IIoT), Internet of Vehicles (IoV), and cloud data sharing.

Mambo et al. [3] first introduced the concept of proxy signatures, enabling an original signer to delegate signing authority to a proxy signer through a warrant-based mechanism, which laid the theoretical foundation for subsequent studies on delegation and security models. With the development of IoT and IIoT applications, Verma et al. [2] proposed an efficient and provable certificate-based proxy signature scheme with low computational overhead and formal security. To reduce certificate management costs, lightweight certificateless signature schemes are proposed for IoT and IIoT systems [21,22]. However, their constructions suffer from key replacement and secret key distribution problems. Identity-based proxy signature schemes are developed to improve data integrity and authentication [23,24]. These schemes rely on centralized key generation centers and involve complex calculations. More proxy signature schemes for IoV and distributed scenarios are proposed based on algebraic or geometric structures (e.g., bilinear pairings, hyperelliptic curves), yet they generally exhibit poor efficiency [9,25,26]. Notably, the security of these schemes fundamentally relies on traditional hardness assumptions (e.g., the discrete logarithm problem), which have been proven vulnerable to quantum cryptanalysis [27,28].

With the threat of quantum computing, lattice-based cryptography emerged as a leading post-quantum candidate. Wang et al. [29] utilized fixed-dimension lattice basis delegation to optimize efficiency for resource-constrained IoT devices, avoiding the key size expansion inherent in previous techniques [30]. To further enhance functionality, Zhu et al. [18] integrated encryption with signing, proposing an identity-based proxy signcryption scheme that ensures both confidentiality and unforgeability. Guo et al. [17] introduced the identity-based linearly homomorphic proxy signature scheme. This scheme allows a third party to perform linear operations on signed messages while maintaining the validity of the signature, facilitating secure data aggregation.

To mitigate the large key overhead of standard lattices, efficient structured variants like NTRU lattices have been adopted. Wu et al. [29] presented an identity-based proxy signature (IBPS) scheme over NTRU lattices. By leveraging the ring structure of polynomial convolutions, their scheme achieves faster signing operations and more compact signatures

compared to unstructured lattice counterparts. Building upon the NTRU framework, Singh et al. [19] recently extended the functionality to blind signatures, preserving the privacy of the message content from the signer.

1.3. Organization

In Section 2, we introduce the notation used throughout the paper and present the necessary preliminaries and basic components. In Section 3, we abstract and model the application scenario and formally define the syntax of our scheme. Section 4 describes the concrete construction of the proposed scheme and provides a security analysis. In Section 5, we present the experimental results and a comparative evaluation with existing schemes. Finally, Sections 6 and 7 conclude the paper and discusses directions for future work.

2. Preliminaries

This section presents some preliminaries of our proposed scheme. We provide an overview of KEM and Dilithium-QROM used in our construction and give the definitions of ARKG and PSUW, two novel frameworks recently proposed in [5,6].

2.1. Notations

Table 1 summarizes the key mathematical notations and conventions used throughout the paper.

Table 1. Notations and descriptions.

Notation	Description
R_q	The polynomial ring $\mathbb{Z}_q[X]/(X^n+1)$
$R_q^{k\times\ell}$	A matrix (or vector) consisting of $k\times\ell$ (or k) elements from R_q
$\mathbf{A}, \mathbf{B}, \mathbf{C}$	Matrices (uppercase bold)
$\mathbf{a}, \mathbf{b}, \mathbf{c}$	Vectors (lowercase bold)
$\mathbf{A}^\top, \mathbf{a}^\top$	Transpose of matrix $\mathbf{A}$ or vector $\mathbf{a}$
$x \leftarrow D$	If D is a distribution, it denotes sampling x from distribution D
$x \leftarrow S$	If S is a set, it denotes choosing x uniformly at random from set S
$\|\cdot\|, \|\cdot\|_\infty$	The ℓ_2 norm and the ℓ_∞ norm of a given element, respectively
B_η	A centered binomial distribution with positive integer parameter η
S_η	The set of all elements satisfying $\|\cdot\|_\infty \leq \eta$
$y \leftarrow \mathrm{A}(x)$	y is sampled from the output of probabilistic algorithm A on input x
$y := \mathrm{A}(x)$	y is the deterministic output of algorithm A on input x
$\mathrm{Sam}(\cdot)$	An extendable output function from $\{0,1\}^* \to \{0,1\}^*$
$H(\cdot)$	A hash function from $\{0,1\}^* \to \{0,1\}^n$ where n is fixed
PPT algorithm	Probabilistic polynomial-time algorithm
ChSet	The set of possible challenges
(pkdS, skdS)	The delegator's key pair for signing
(pkpS, skpS)	The proxy's key pair for signing

2.2. Dilithium-QROM Signature

Dilithium, which is based on the Module-LWE and Module-SIS problems, has been standardized by NIST as the post-quantum cryptography algorithm ML-DSA [14]. To achieve a tight security reduction in the quantum random oracle model (QROM), Kiltz et al. proposed Dilithium-QROM [16], a variant of Dilithium that fundamentally relies on the hardness of the MLWE problem. This scheme introduces the concept of lossy public keys and constructs a modular security framework, thereby providing resistance against quantum attacks.

Compared to the standard Dilithium scheme, to support a lossy security proof, Dilithium-QROM employs larger parameters n and q, modifies the challenge generation procedure for c, and introduces an explicit upper bound on the counter in the rejection

sampling loop. Unlike Dilithium, Dilithium-QROM has only a single bound parameter γ. While these modifications enable a tight security reduction, they also incur increased storage overhead. The Dilithium-QROM adopted in this work is built upon the construction proposed by Eaton et al. [31]. We reorganize its algorithmic structure to conform to the classical three-tuple signature paradigm (KeyGen, Sign, Verify) without modifying the underlying hardness assumptions, security guarantees, or computational correctness. We give a general description of Dilithium-QROM in Figure 1.

KeyGen()

$\rho, K \leftarrow \{0,1\}^{256}$
$\mathbf{A} \leftarrow R_q^{k\times\ell} := \mathrm{Sam}(\rho)$
$(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \mathcal{S}_\eta^\ell \times \mathcal{S}_\eta^k$
$\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
$\mathbf{t}_1 := \mathrm{Power2Round}_q(\mathbf{t}, d)$
$\mathbf{t}_0 := \mathbf{t} - \mathbf{t}_1 \cdot 2^d$
$\mathrm{pk} = (\rho, \mathbf{t}_1)$
$\mathrm{sk} = (\rho, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0, K)$
return $(\mathrm{pk}, \mathrm{sk})$

Sign(sk, M)

$\kappa := 0$
$\mathbf{A} \leftarrow R_q^{k\times\ell} := \mathrm{Sam}(\rho)$
while $(\mathbf{z}, \mathbf{h}) = \perp$ and $\kappa \leq 200/(1-\delta)$
  $\kappa := \kappa + 1$
  $\mathbf{y} \leftarrow S_{\gamma-1}^\ell := \mathrm{Sam}(K \| M \| \kappa)$
  $\mathbf{w} := \mathbf{A}\mathbf{y}$
  $\mathbf{w}_1 := \mathrm{HighBits}_q(\mathbf{w}, 2\gamma)$
  $c := \mathrm{H}(\mathbf{w}_1 \| M)$
  $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
  if $\|\mathbf{z}\|_\infty \geq \gamma - \beta$ or $\|\mathrm{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma)\|_\infty \geq \gamma - \beta$
    $(\mathbf{z}, \mathbf{h}) := \perp$
  else
    $\mathbf{h} := \mathrm{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma)$
return $\sigma = (\mathbf{z}, \mathbf{h}, c)$

Ver(pk, $M, \sigma = (\mathbf{z}, \mathbf{h}, c)$)

$\mathbf{A} \leftarrow R_q^{k\times\ell} := \mathrm{Sam}(\rho)$
$\mathbf{w}_1' := \mathrm{UseHint}_q(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, 2\gamma)$
return $[\|\mathbf{z}\|_\infty < \gamma - \beta]$ and $[c = \mathrm{H}(\mathbf{w}_1' \| M)]$

Figure 1. KeyGen, Sign and Verify algorithms of Dilithium-QROM. The detailed implementations of supporting algorithms, including Power2Round, MakeHint, UseHint, Decompose and more, can be found in the specification of Dilithium [14].

2.3. CRYSTALS-Kyber

Kyber [11] is a key encapsulation mechanism (KEM) based on the MLWE problem and has been standardized as Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM) [13]. It is constructed by applying the Fujisaki–Okamoto (FO) transform [32] to an IND-CPA-secure public-key encryption scheme, thereby upgrading its security to IND-CCA2. We briefly describe KyberKEM = (KeyGen, EnCap, DeCap) as follows.

- KyberKEM. $\mathrm{KeyGen}(1^\lambda) \rightarrow (\mathrm{pk}_{kem}, \mathrm{sk}_{kem})$: The probabilistic key generation algorithm takes the security parameter λ as input and outputs a public-secret key pair $(\mathrm{pk}_{kem}, \mathrm{sk}_{kem})$ where $\mathrm{pk}_{kem} \in \{0,1\}^{256} \times R_{q_k}^k$ and $\mathrm{sk}_{kem} \in R_{q_k}^k$.
- KyberKEM. $\mathrm{Encap}(\mathrm{pk}_{kem}) \rightarrow (K, \mathbf{c})$: The probabilistic encapsulation algorithm takes as input a encapsulation public key pk_{kem} and outputs a shared secret $K \in \{0,1\}^{256}$ and a ciphertext $\mathbf{c} \in R_{q_k}^{k+1}$.

- KyberKEM.Decap(sk_{kem}, **c**) $\rightarrow K$ or $\perp$: The deterministic decapsulation algorithm takes as input a decapsulation private key sk_{kem} and the ciphertext **c** and outputs a shared secret K or $\perp$ on failure.

The Kyber specification provides three parameter sets corresponding to different security levels. Their security levels are based on concrete complexity estimates for BKZ lattice reduction across varying block sizes [33–35]. On the theoretical side, KyberKEM is rigorously proven to be IND-CCA secure in the QROM under the MLWE assumption [36].

In this work, we use the Kyber-512 as a building block of our proposed scheme, and denote it as KyberKEM throughout.

2.4. Asynchronous Remote Key Generation, ARKG

ARKG is a recently proposed cryptographic primitive that enables the asynchronous derivation of fresh, unlinkable public keys from an original public key [5]. Only the legitimate key owner can derive the corresponding private keys using the original private key together with auxiliary credentials. Its core security properties are public-key unlinkability and private-key unforgeability.

Definition 1 (Asynchronous Remote Key Generation)**.** *The remote key generation and recovery scheme* $\mathrm{ARKG} = (\mathrm{Setup}, \mathrm{KGen}, \mathrm{DerivePK}, \mathrm{DeriveSK}, \mathrm{Check})$ *consists of the following PPT algorithms:*

- $\mathrm{Setup}(\lambda)$*: Generate and output public parameters* pp *of the scheme for the security parameter* $\lambda \in \mathbb{N}$*.*
- $\mathrm{KGen}(\mathrm{pp})$*: Given input parameters* pp*, generate a private–public key pair* $(\mathrm{pk}, \mathrm{sk})$*.*
- $\mathrm{DerivePK}(\mathrm{pk}, \mathrm{aux})$*: Return a new public key* pk' *together with a credential* cred *linking* pk *and* pk'*. The input auxiliary information* aux *is always required but may be empty.*
- $\mathrm{DeriveSK}(\mathrm{sk}, \mathrm{cred})$*: Output either the new private key* sk'*, corresponding to* pk' *via* cred*, or* $\perp$ *on error.*
- $\mathrm{Check}(\mathrm{pk}', \mathrm{sk}')$*: On input* $(\mathrm{pk}', \mathrm{sk}')$*, return* true *if* $(\mathrm{pk}', \mathrm{sk}')$ *forms a valid private–public key pair and false otherwise.*

2.5. Proxy Signatures with Unlinkable Warrants

Proxy-Signatures with Unlinkable Warrants (PSUW) [6] is built upon ARKG and was originally designed to enable secure account delegation while preserving the core privacy guarantees of the WebAuthn standard. The formal definition of PSUW is presented below.

Definition 2 (Proxy-Signatures with Unlinkable Warrants)**.** *The PSUW scheme consists of six PPT algorithms:*

- $\mathrm{Setup}(\lambda)$*: Generate public parameters* pp *for security parameter* $\lambda \in \mathbb{N}$*.*
- $\mathrm{dKeyGen}(\mathrm{pp})$*: Generate private–public key pair* $(\mathrm{pkd}, \mathrm{skd})$ *for a delegator.*
- $\mathrm{pKeyGen}(\mathrm{pp})$*: Generate private–public key pair* $(\mathrm{pkp}, \mathrm{skp})$ *for a proxy.*
- $\mathrm{Delegate}(\mathrm{pkd}, \mathrm{skp})$*: Take the proxy's public key* pkp *and the delegator's private key* skd *as input, and probabilistically return a warrant* warr *and delegation data* ddata*.*
- $\mathrm{Sign}(\mathrm{skp}, \mathrm{pkd}, \mathrm{warr}, \mathrm{ddata}, m)$*: Given the proxy's private key* skp*, the delegator's public key* pkd*, warrant* warr*, delegation data* ddata*, and message* m*, return a proxy signature* $\bar{\sigma}$*, or* $\perp$ *on error. Note that* $\mathrm{warr} \in \bar{\sigma}$*, allowing* $\bar{\sigma}$ *to be verified as a standalone signature. The* ddata *is used to compute the signing key but is not included in* $\bar{\sigma}$*.*
- $\mathrm{Verify}(\mathrm{pkd}, \bar{\sigma}, m)$*: Return* true *if* $\bar{\sigma}$ *is a valid proxy signature on* m *with respect to* pkd*, and* false *otherwise.*

3. Problem Formulation

In this section, we first describe the workflow of proxy signatures in IoT scenarios and define the involved entities Section 3.1. We then present the syntax of our scheme in detail in Section 3.2. Finally, we formalize the corresponding security definitions in Section 3.3.

3.1. System Model of IoT Proposed Scheme

In Internet of Things (IoT) environments, delegating signing capabilities to a proxy entity typically arises in two common situations: the original signer (e.g., the device owner) may be temporarily unavailable to participate in authentication procedures or IoT devices often have limited computational resources and cannot efficiently handle large volumes of authentication or signing requests.

In both cases, the proxy entity performs signing operations on behalf of the original signer within the scope defined by a delegation credential while allowing external parties to verify the validity of the proxy-generated signatures. Based on these observations, we integrate these IoT delegation scenarios into a three-entity model as shown in Figure 2, in which an IoT device or its data owner delegates signing capabilities to a more available or computationally powerful entity to perform authentication or signing tasks on their behalf.

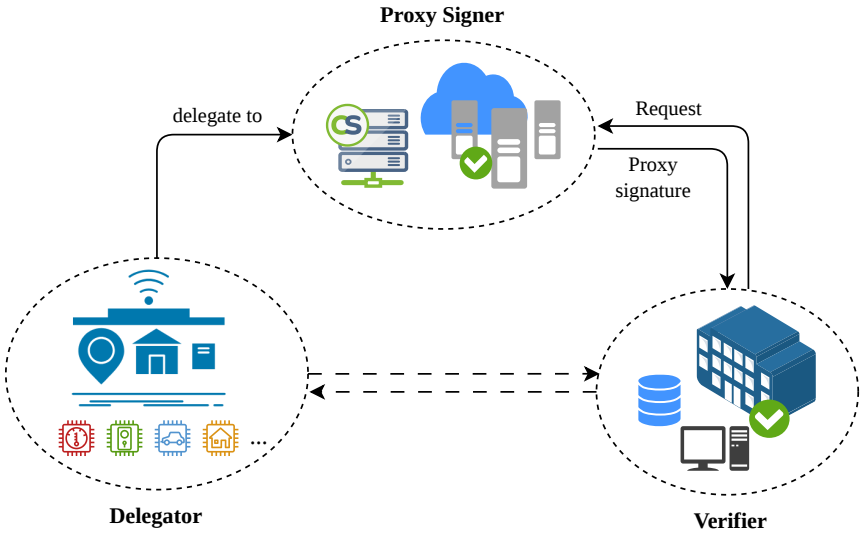

Figure 2. System model of IoT environment using delegation.

- **Delegator:** The delegator is typically an IoT device (e.g., a sensor or controller) or its owner, who possesses the original signing key pair and controls the device or its data. The delegator authorizes a proxy signer by issuing a verifiable delegation credential.
- **Proxy Signer:** The proxy signer is the core entity in the proxy signature process. In IoT scenarios, it is usually a trusted entity with stronger computational capabilities, such as a fog node or a cloud server. The proxy signer holds its own long-term key pair and, after receiving authorization from the delegator, obtains the capability to generate proxy signatures on behalf of the delegator.
- **Verifier:** The verifier represents external service providers or authentication servers, such as device manufacturers or data processing centers. In the system model, the verifier primarily interacts with the proxy signer but may also communicate with the delegator when necessary. Its role is to verify both the legitimacy of the delegation credential and the correctness of the proxy signature generated by the proxy signer.

Remark 1. *As illustrated in the system model, the delegator typically consists of low-power IoT terminals. To conserve energy, these devices frequently enter sleep mode, making real-time synchronization with the proxy signer impractical; thus, an asynchronous authorization mechanism is essential. Furthermore, the signing workload can be delegated to entities with stronger computational capabilities. Benefiting from Dilithium's trapdoor-free design, DPS achieves a notable speedup over trapdoor-based schemes, as validated by our experiments, and supports flexible deployment on resource-constrained IoT nodes.*

3.2. Syntax of Proposed Scheme

We propose a Dilithium-based proxy signature (DPS) scheme instantiated under the ARKG and PSUW frameworks. The proposed ARKG is a 5-tuple based on the Kyber KEM, consisting of the following PPT algorithms: (SetUp, KeyGen, DerivePK, DeriveSK, Check).

1. ARKG. $\mathrm{SetUp}(\lambda) \rightarrow \mathrm{pp}$: On input the security parameter λ, this algorithm initializes the global system parameters and outputs two public parameter sets: ppK for the key encapsulation component and ppS for the signature component. The complete public parameter tuple is defined as $\mathrm{pp} = (\mathrm{ppK}, \mathrm{ppS})$.
2. ARKG. $\mathrm{KeyGen}(\mathrm{pp}) \rightarrow (\mathrm{pk}_{kem}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS})$: This algorithm invokes Kyber KEM. KeyGen with ppK and Dilithium-QROM. KeyGen with pp, obtaining the corresponding keys $(\mathrm{pk}_{kem}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS})$, where $(\mathrm{pk}_{kem}, \mathrm{sk}_{kem})$ is the encapsulation key pair and $(\mathrm{pkpS}, \mathrm{skpS})$ is the proxy's signature key pair.
3. ARKG. $\mathrm{DerivePK}(\mathrm{pkpS}, \mathrm{pk}_{kem}) \rightarrow (\mathrm{pk}_{\tau}, \mathrm{cred})$: This algorithm, executed by the delegator, takes as input the proxy's encapsulation public key pk_{kem} and the proxy's signature public key pkpS, outputs the derived public key pk_{τ} for pkpS and credential cred.
4. ARKG. $\mathrm{DeriveSK}(\mathrm{cred}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS}) \rightarrow \mathrm{sk}_{\tau}$: This algorithm, executed by the proxy, takes as input the credential cred transmitted by the delegator, the decapsulation private key sk_{kem}, and its signature key pair pkpS, skpS, and outputs the derived signature private key sk_{τ}.
5. ARKG. $\mathrm{Check}(\mathrm{pk}_{\tau}, \mathrm{sk}_{\tau})$: This algorithm returns true if pk_{τ} and sk_{τ} form a valid public–private key pair, and false otherwise.

We treat ARKG as an encapsulated cryptographic component and propose the DPS scheme based on the PSUW framework, which consists of six PPT algorithms as described below.

1. DPS. $\mathrm{SetUp}(\lambda) \rightarrow \mathrm{pp}$: This algorithm initializes the global environment of the delegation system. Given the security parameter λ, it outputs public parameters pp, including ppK for the key encapsulation component and ppS for the signature component.
2. In the key generation phase, the algorithm generates long-term identity keys. Since the delegator and the proxy assume different roles within the protocol, we distinguish between two key generation procedures. Specifically, given the public parameters pp, the algorithms are defined as follows.
   - DPS. $\mathrm{dKeyGen}(\mathrm{ppS}) \rightarrow (\mathrm{pkdS}, \mathrm{skdS})$: The key generation procedure for the delegator is denoted by DPS. dKeyGen. On inputting public parameter ppS, the delegator invokes Dilithium-QROM. KeyGen to generate its signature key pairs $(\mathrm{pkdS}, \mathrm{skdS})$.
   - DPS. $\mathrm{pKeyGen}(\mathrm{pp} = (\mathrm{ppK}, \mathrm{ppS})) \rightarrow (\mathrm{pk}_{kem}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS})$: The key generation procedure for the proxy signer is denoted by DPS. pKeyGen. On inputting public parameter ppK and ppS, the proxy signer invokes ARKG. KeyGen to generate its key pairs for encapsulation and signature respectively.

https://doi.org/10.3390/cryptography10030033

3. DPS. Delegate$(\mathrm{pkpS}, \mathrm{pk}_{kem}, \mathrm{skdS}) \rightarrow (\mathrm{ddata}, \mathrm{warr} = (\mathrm{pk}_\tau, \sigma_\tau))$: Taking as input the proxy's signature public key pkpS, encapsulation public key pk_{kem} and the delegator's signing key skdS, this algorithm invokes ARKG. DerivePK$(\mathrm{pkpS}, \mathrm{pk}_{kem})$ to obtain a derived public key pk_τ and credential cred, where cred is packed into ddata for the proxy to recover the corresponding signing key sk_τ. The delegator then obtains the signature σ_τ by signing the derived public key pk_τ with skdS, forming the warrant $\mathrm{warr} := (\mathrm{pk}_\tau, \sigma_\tau)$ that cryptographically binds pk_τ to the delegator's identity and authorizes the proxy to sign on the delegator's behalf, and returns $(\mathrm{ddata}, \mathrm{warr})$.
4. DPS. ProxySign$(\mathrm{ddata}, \mathrm{pkdS}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS}, \mathrm{warr}, m) \rightarrow \sigma_m$: After verifying the warrant warr, the proxy signing algorithm first invokes ARKG. DeriveSK with the credential cred extracted from ddata and decapsulation key sk_{kem} and proxy's key pair $(\mathrm{pkpS}, \mathrm{skpS})$ to derive the proxy signing key sk_τ, then signs message m with sk_τ to output a valid proxy signature σ_m using Dilithium-QROM. Sign.
5. DPS. Verify$(\mathrm{pkdS}, \mathrm{warr}, \sigma_m, m) \rightarrow \{\mathrm{true}, \mathrm{false}\}$: The verification algorithm is executed by a third-party verification platform and aims to confirm that a signature has been generated by an authorized proxy. Given the delegator's signature public key pkdS, the corresponding authorization $\mathrm{warr} = (\mathrm{pk}_\tau, \sigma_\tau)$, the message attached with its proxy signature σ_m, the algorithm verifies signature σ_τ with pkdS and signature σ_m with pk_τ. It outputs true if both verifications pass, and false otherwise.

3.3. Security Definitions

The DPS scheme satisfies the security properties of proxy signatures [37], which are described as follows:

- Unforgeability: A designated proxy signer can create a valid proxy signature for the original signer. But the original signer and other third parties who are not designated as a proxy signer cannot create a valid proxy signature.
- Distinguishability: Proxy signatures are distinguishable from normal signatures by everyone.
- Verifiability: From the proxy signature, the verifier can be convinced of the original signer's agreement on the signed message.
- Strong undeniability: Once a proxy signer creates a valid proxy signature of an original signer, he/she cannot repudiate the signature creation.

To satisfy the unforgeability property, an adversary $\mathcal{A}$ should not be able to forge a proxy signature with respect to a delegator's public key pkdS without knowledge of the corresponding secret key skdS. Based on the PSUW framework [7], we define the unforgeability experiment for our scheme in Figure 3. The adversary $\mathcal{A}$ is allowed to make a polynomial number of queries to the oracles modeling registration, delegation, proxy signing, and corruption, which are formally defined in Appendix A.

$\mathrm{Exp}^{\mathrm{Unforge}}_{DPS,\mathcal{A}}(\lambda)$

1: $\mathrm{pp} \leftarrow \mathrm{Setup}(\lambda)$
2: $(\mathrm{skdS}, \mathrm{pkdS}) \xleftarrow{\$} \mathrm{DPS.dKeyGen}()$
3: $(\bar{\sigma}, m) \xleftarrow{\$} \mathcal{A}^{\mathcal{O}_{\mathrm{Reg}}, \mathcal{O}_{\mathrm{Delegate}}, \mathcal{O}_{\mathrm{ProxySign}}, \mathcal{O}_{\mathrm{Corr}}}(\mathrm{pp}, \mathrm{pkdS})$
4: parse $\bar{\sigma}$ as $(\mathrm{warr}, \cdot)$
5: **return** $\mathrm{DPS.Verify}(\mathrm{pkdS}, \bar{\sigma}, m) = 1 \wedge$
6: $\quad\big($ $(\cdot, \mathrm{warr}) \notin \mathcal{L}_{\mathrm{Del}} \vee$
7: $\quad\big[\exists \mathrm{pkpS}, m$ s.t. $(\cdot, \cdot, \mathrm{warr}, \cdot, m) \notin \mathcal{L}_{\mathrm{Sign}} \wedge (\mathrm{pkpS}, \mathrm{warr}) \in \mathcal{L}_{\mathrm{Del}} \wedge$
8: $\quad(\cdot, \mathrm{pkpS}) \notin \mathcal{L}_{\mathrm{Corrupt}}\big]\big)$

Figure 3. Unforgeability experiment for DPS scheme.

Definition 3 (Unforgeability). *Let $\mathcal{A}$ be any probabilistic polynomial-time adversary against the unforgeability of the proposed scheme with security parameter λ. A DPS scheme provides unforgeability if the following advantage is negligible in λ:*

$$\mathrm{Adv}^{unforge}_{DPS,\mathcal{A}}(\lambda) := \Pr\left[\mathrm{Exp}^{unforge}_{DPS,\mathcal{A}}(\lambda) = 1\right]$$

In addition, DPS provides an unlinkability property by using the PSUW framework. This property is described as follows:

- Unlinkability: If an adversary observes multiple derived public keys and their corresponding proxy signatures, it cannot link them to the long-term public key of the proxy signer.

4. Proposed Scheme

In this section, we first present an ARKG scheme based on Kyber KEM in Section 4.1. Building upon this ARKG scheme and the PSUW framework, we propose a post-quantum proxy signature scheme in the QROM based on Dilithium in Section 4.2. Finally, we provide the security proof and correctness proof of our scheme in Section 4.3 and Section 4.4, respectively.

4.1. ARKG Instantiated on KyberKEM

Inspired by [31], we have instantiated the ARKG scheme based on Kyber KEM, and in this instance, the derived public and private keys can serve as input values for the Dilithium signature algorithm.

1. ARKG.SetUp(λ) : Given the security parameters $\lambda \in \mathbb{N}$, the system initializes and generates the system public parameters and hash functions using the parameter setting method of the lattice cryptography. The specific algorithm is below.
   (1) Runs KyberKEM. SetUp(λ) to get parameter ppK.
   (2) Runs Dilithium-QROM. SetUp(λ) to get parameters $(q_s, n_s, k, l, d, d', \beta, \gamma, \eta_s)$, where q_s and n_s define the polynomial ring $R = \mathbb{Z}_{q_s}[X]/(X^{n_s}+1)$, k and l determine the matrix dimensions, d and d' are rounding parameters, and β, γ, and η_s specify the bounds used in key generation and signing.
   (3) Initializes secure functions: Hash function $H_1 : \{0,1\}^* \times R^k_{q_s} \times R^k_{q_s} \to$ ChSet, Extendable output function(XOF) Sam : $\{0,1\}^* \to R^{k\times\ell}_{q_s}$, Deterministic sampling algorithms based on XOF $G_1 : R^\ell_{q_s} \times \{0,1\}^* \to S^\ell_{\eta_s} \times S^k_{\eta_s}$, and $G_2 : \{0,1\}^* \to S^\ell_{\gamma-1}$.
   (4) Publishes the public parameters pp = (ppK, ppS), where ppS = $(q_s, n_s, k, l, d, d', \beta, \gamma, \eta_s, H_1, \mathrm{Sam}, G_1, G_2)$.
2. ARKG. KeyGen(pp): Taking the public parameter pp as input, it computes and outputs $(\mathrm{pk}_{kem}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS})$ as described in Algorithm 1, where $(\mathrm{pk}_{kem}, \mathrm{sk}_{kem})$ is the encapsulation key pair and $(\mathrm{pkpS}, \mathrm{skpS})$ is the proxy's signature key pair.
3. ARKG.DerivePK($\mathrm{pkpS}, \mathrm{pk}_{kem}$): Given the proxy's signature public key pkpS and the encapsulation public key pk_{kem}, it computes and outputs $(\mathrm{pk}_\tau, \mathrm{cred})$ as described in Algorithm 2, where pk_τ is the derived key for pkpS and cred is a credential data which contains the link between the original key and the derived key.
4. ARKG.DeriveSK($\mathrm{cred}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS}$): Given the credential cred, the decapsulation private key sk_{kem}, and the proxy's signature key pair $(\mathrm{pkpS}, \mathrm{skpS})$, it computes and outputs the derived secret key sk_τ for skpS as described in Algorithm 3.
5. ARKG.Check($\mathrm{pk}_\tau, \mathrm{sk}_\tau$): Given the derived key pair pk_τ and sk_τ, this algorithm outputs a boolean value as described in Algorithm 4.

Algorithm 1 ARKG.KeyGen(pp)

Input: Public parameter $\mathrm{pp} = (\mathrm{ppK}, \mathrm{ppS})$.
Output: An encapsulation key pair $(\mathrm{sk}_{kem}, \mathrm{pk}_{kem})$ and a signature key pair$(\mathrm{pkpS}, \mathrm{skpS})$.

$(\mathrm{pk}_{kem}, \mathrm{sk}_{kem}) \leftarrow \mathrm{KyberKEM.KeyGen}(\mathrm{ppK})$
$\rho_{s,p}, K_p \leftarrow \{0,1\}^{256}$
$\mathbf{A} \leftarrow R_{q_s}^{k \times \ell} := \mathrm{Sam}(\rho_{s,p})$
$(\mathbf{s}_{1,p}, \mathbf{s}_{2,p}) \leftarrow S_{\eta_s}^{\ell} \times S_{\eta_s}^{k}$
$\mathbf{t}_p := \mathbf{A}\mathbf{s}_{1,p} + \mathbf{s}_{2,p}$
$\mathbf{t}_{1,p} := \mathrm{Power2Round}_{q_s}(\mathbf{t}_p, d)$
$\bar{\mathbf{t}}_{1,p} := \mathrm{Power2Round}_{q_s}(\mathbf{t}_p, d' - 1)$
$\mathbf{t}_{0,p} := \mathbf{t}_p - \mathbf{t}_{1,p} \cdot 2^d$
$\mathrm{pkpS} = (\rho_{s,p}, \mathbf{t}_{1,p}, \bar{\mathbf{t}}_{1,p})$
$\mathrm{skpS} = (\rho_{s,p}, \mathbf{s}_{1,p}, \mathbf{s}_{2,p}, \mathbf{t}_{0,p}, K_p)$
return $(\mathrm{pk}_{kem}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS})$

Algorithm 2 ARKG.DerivePK(pkpS, pk_{kem})

Input: Proxy's signature public key $\mathrm{pkpS} = (\rho_{s,p}, \mathbf{t}_{1,p}, \bar{\mathbf{t}}_{1,p})$ and the encapsulation public key pk_{kem}.
Output: The derived public key pk_τ, and credential data cred.

$(\tau, \mathbf{v}) \leftarrow \mathrm{KyberKEM.Encap}(\mathrm{pk}_{kem})$
$(\mathbf{s}'_{1,p}, \mathbf{s}'_{2,p}) := \mathrm{G}_1(\bar{\mathbf{t}}_{1,p} \| \tau)$
$\mathbf{A} \leftarrow \mathrm{Sam}(\rho_{s,p})$
$\mathbf{t}'_p := \mathbf{A}\mathbf{s}'_{1,p} + \mathbf{s}'_{2,p}$
$\mathbf{t}'_{1,p} := \mathrm{Power2Round}_{\mathrm{q_s}}(\mathbf{t}'_p, d' - 1)$
$\bar{\mathbf{t}}_{1,\tau} := \bar{\mathbf{t}}_{1,p} + \mathbf{t}'_{1,p}$
return $(\mathrm{pk}_\tau = (\rho_{s,p}, \bar{\mathbf{t}}_{1,\tau}), \mathrm{cred} = \mathbf{v})$

Algorithm 3 ARKG.DeriveSK(cred, sk_{kem}, pkpS, skpS)

Input: The credential $\mathrm{cred} = \mathbf{v}$, the decapsulation private key sk_{kem}, and the proxy's signature key pair $(\mathrm{pkpS}, \mathrm{skpS}) = ((\rho_{s,p}, \mathbf{t}_{1,p}, \bar{\mathbf{t}}_{1,p}), (\rho_{s,p}, \mathbf{s}_{1,p}, \mathbf{s}_{2,p}, \mathbf{t}_{0,p}, K_p))$.
Output: The derived secret key sk_τ.

$\tau := \mathrm{KyberKEM.Decap}(\mathrm{sk}_{kem}, \mathbf{v})$
$(\mathbf{s}'_{1,p}, \mathbf{s}'_{2,p}) \leftarrow \mathrm{G}_1(\bar{\mathbf{t}}_{1,p} \| \tau)$
$\mathbf{s}_{1,\tau} := \mathbf{s}'_{1,p} + \mathbf{s}_{1,p}$
$\mathbf{s}_{2,\tau} := \mathbf{s}'_{2,p} + \mathbf{s}_{2,p}$
$\mathbf{A} \leftarrow \mathrm{Sam}(\rho_{s,p})$
$\mathbf{t}_\tau := \mathbf{A}\mathbf{s}_{1,\tau} + \mathbf{s}_{2,\tau}$
$\mathbf{t}_{0,\tau} := \mathbf{t}_\tau - \mathrm{Power2Round}_{q_s}(\mathbf{t}_\tau, d')$
$K_\tau := K_p$
return $\mathrm{sk}_\tau = (\rho_{s,p}, \mathbf{s}_{1,\tau}, \mathbf{s}_{2,\tau}, \mathbf{t}_{0,\tau}, K_\tau)$

Algorithm 4 ARKG.Check(pk_τ, sk_τ)

Input: The derived key pair $\mathrm{pk}_\tau = (\rho_{s,p}, \bar{\mathbf{t}}_{1,\tau})$ and $\mathrm{sk}_\tau = (\rho_{s,p}, \mathbf{s}_{1,\tau}, \mathbf{s}_{2,\tau}, \mathbf{t}_{0,\tau}, K_\tau)$.
Output: If the check passes, it outputs true; otherwise, false.

$\mathbf{A} \leftarrow \mathrm{Sam}(\rho_{s,p})$
$\mathbf{t}_\tau := \mathbf{A}\mathbf{s}_{1,\tau} + \mathbf{s}_{2,\tau}$
$\mathbf{t}_{1,\tau} := \mathrm{Power2Round}_{q_s}(\mathbf{t}_\tau, d')$
$\alpha_1 := \mathrm{Power2Round}_{q_s}(2\mathbf{t}_{1,\tau} \cdot 2^{d'-1}, d')$
$\alpha_2 := \mathrm{Power2Round}_{q_s}(\bar{\mathbf{t}}_{1,\tau} \cdot 2^{d'-1}, d')$
if $\|\alpha_1 - \alpha_2\|_\infty \leq 1$ **then**
    return true
end if
return false

4.2. Description of Proposed DPS Scheme

Based on the above building blocks, we propose an innovative lattice-based proxy signature scheme DPS, which relies on our proposed ARKG scheme and the PSUW framework. We also give the workflow of our scheme as shown in Figure 4.

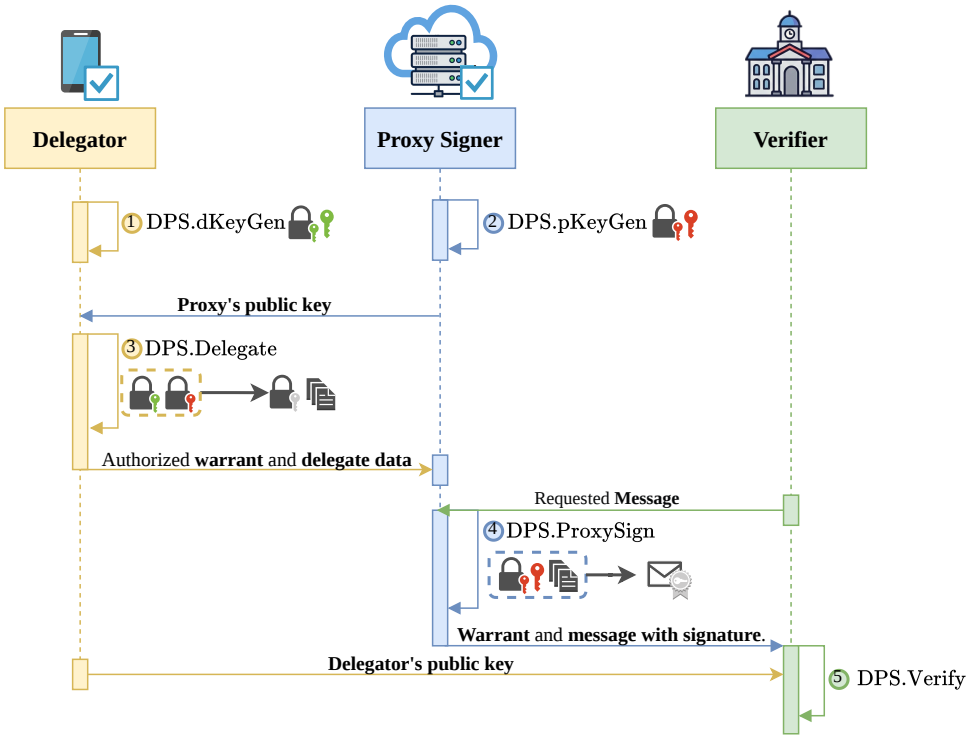

Figure 4. The workflow of DPS scheme.

1. DPS.Setup(λ): Given the security parameters λ, the system runs ARKG.SetUp(λ) to get public parameters pp, and parse it as (ppK, ppS).
2. DPS.dKeyGen(ppS): Delegator generates its own key pair by running Dilithium-QROM.KeyGen(ppS) to generate $(\text{pkdS}, \text{skdS})$:

$$\begin{aligned}\text{pkdS} &= (\rho_{s,d}, \mathbf{t}_{1,d}) \in \{0,1\}^{256} \times R_{q_s}^k,\\ \text{skdS} &= (\rho_{s,d}, \mathbf{s}_{1,d}, \mathbf{s}_{2,d}, \mathbf{t}_{0,d}, K_d) \in \{0,1\}^{256} \times S_{\eta_s}^{\ell} \times S_{\eta_s}^{k} \times R_{q_s}^{k} \times \{0,1\}^{256}.\end{aligned}$$

3. DPS.pKeyGen(pp): The proxy signer generates its own key pairs by running ARKG.KeyGen(pp) to generate $(\text{pk}_{kem}, \text{sk}_{kem})$ and $(\text{skpS}, \text{pkpS})$:

$$\begin{aligned}\text{pkpS} &= (\rho_{s,p}, \mathbf{t}_{1,p}, \bar{\mathbf{t}}_{1,p}) \in \{0,1\}^{256} \times R_{q_s}^k \times R_{q_s}^k,\\ \text{skpS} &= (\rho_{s,p}, \mathbf{s}_{1,p}, \mathbf{s}_{2,p}, \mathbf{t}_{0,p}, K_p) \in \{0,1\}^{256} \times S_{\eta_s}^{\ell} \times S_{\eta_s}^{k} \times R_{q_s}^{k} \times \{0,1\}^{256}.\end{aligned}$$

4. DPS.Delegate$(\text{pkpS}, \text{pk}_{kem}, \text{skdS})$: This algorithm takes as input the proxy's signature public key pkpS, encapsulation public key pk_{kem}, and the delegator's signature private key skdS. It computes and outputs the derived public key pk_{τ} for pkpS, the delegator's signature σ_{τ} on this derived public key, and the delegation data ddata as auxiliary information. The formal procedure is described as follows.

   (1) Given the proxy's public key $\text{pkpS} = (\rho_{s,p}, \mathbf{t}_{1,p}, \bar{\mathbf{t}}_{1,p})$ and pk_{kem}, compute

$$(\text{pk}_{\tau}, \text{cred}) \leftarrow \text{ARKG.DerivePK}(\text{pkpS}, \text{pk}_{kem}).$$

(2) Compute $\sigma_\tau \leftarrow$ Dilithium-QROM.sign$(\text{skdS}, \text{pk}_\tau)$, and output (ddata = cred, warr = $(\text{pk}_\tau, \sigma_\tau)$).

5. DPS.ProxySign$(\text{ddata}, \text{pkdS}, \text{sk}_{kem}, \text{pkpS}, \text{skpS}, \text{warr}, m)$: The algorithm first verifies the warrant warr under the delegator's public key pkdS. If the verification fails, it outputs $\perp$. Otherwise, it invokes ARKG.DeriveSK on ddata, sk_{kem}, and the proxy signing key pair (pkpS, skpS) to derive the delegated signing key. It then uses the derived key to sign m and outputs the proxy signature (σ_m, warr). The detailed procedure is given in Algorithm 5.

Algorithm 5 DPS.ProxySign$(\text{ddata}, \text{pkdS}, \text{sk}_{kem}, \text{pkpS}, \text{skpS}, \text{warr}, m))$

Input: The delegation data ddata, the delegator's public key *pkdS*, the decapsulation private key sk_{kem}, the proxy's signature key pair (pkpS, skpS), the warrant warr = $(\text{pk}_\tau, \sigma_\tau)$, and the message m.
Output: The signature σ_m for m and the warrant warr.

if Dilithium-QROM.Verify$(\text{pkdS}, \text{pk}_\tau, \sigma_\tau)$ = false **then return** $\perp$
end if
$(\rho_{s,p}, \mathbf{s}_{1,\tau}, \mathbf{s}_{2,\tau}, \mathbf{t}_{0,\tau}, K_\tau) \leftarrow$ ARKG.DeriveSK$(\text{ddata}, \text{sk}_{kem}, \text{pkpS}, \text{skpS})$
$\mathbf{A} \leftarrow R_q^{k\times\ell} := \text{Sam}(\rho_{s,p})$
$\kappa := 0$
while $(\mathbf{z}, \mathbf{h}) = (\perp, \perp)$ **and** $\kappa \le 200/(1-\delta)$ **do**
  $\kappa \leftarrow \kappa + 1$
  $\mathbf{y} \leftarrow \mathcal{S}_{\gamma-1}^{\ell} := G_2(K_\tau \| m \| \kappa)$
  $\mathbf{w} := \mathbf{A}\mathbf{y}$
  $\mathbf{w}_1 := \text{HighBits}_{q_s}(\mathbf{w}, 2\gamma)$
  $c \leftarrow \text{H}_1(m \| \mathbf{w}_1 \| \bar{\mathbf{t}}_{1,\tau})$
  $\mathbf{z} := \mathbf{y} + c\mathbf{s}_{1,\tau}$
  if $\|\mathbf{z}\|_\infty \ge \gamma - 2\beta$ or $\| \text{LowBits}_{q_s}(\mathbf{w} - c\mathbf{s}_{2,\tau}, 2\gamma)\|_\infty \ge \gamma - 2\beta$ **then**
    $(\mathbf{z}, \mathbf{h}) \leftarrow (\perp, \perp)$
  else
    $\mathbf{h} := \text{MakeHint}_{q_s}(-c\mathbf{t}_{0,\tau}, \mathbf{w} - c\mathbf{s}_{2,\tau} + c\mathbf{t}_{0,\tau}, 2\gamma)$
  end if
end while
$\sigma_m := (\mathbf{z}, \mathbf{h}, c)$
return (σ_m, warr)

6. DPS.Verify$(\text{pkdS}, \text{warr}, \sigma_m, m)$: This verification algorithm takes as input the delegator's signature public key pkdS, the warrant authorized by delegator warr = $(\text{pk}_\tau, \sigma_\tau)$, a proxy signature σ_m and the message m. It checks whether the signature σ_m is valid under the public key pk_τ, and simultaneously verifies whether the warrant warr = $(\text{pk}_\tau, \sigma_\tau)$ is valid using the verification function Dilithium-QROM.Verify$(\text{pkdS}, \text{pk}_\tau, \sigma_\tau)$. The procedure is formally described in Algorithm 6.

Algorithm 6 DPS.Verify$(\text{pkdS}, \text{warr}, \sigma_m, m)$

Input: The delegator's public key pkdS, the signature on requested message $\sigma_m = (\mathbf{z}_m, \mathbf{h}_m, c_m)$, the warrant warr, and the requested message m.
Output: The algorithm outputs true if both verifications succeed, and false otherwise.

$\mathbf{A} \leftarrow \text{Sam}(\rho_{s,p})$
$\mathbf{w}_1' := \text{UseHint}_{q_s}(\mathbf{h}_m, \mathbf{A}\mathbf{z}_m - c_m\bar{\mathbf{t}}_{1,\tau} \cdot 2^{d'-1}, 2\gamma)$
if $\|\mathbf{z}_m\|_\infty < \gamma - 2\beta$ **and** $c_m = \text{H}_1(m \| \mathbf{w}_1' \| \bar{\mathbf{t}}_{1,\tau})$ **and** Dilithium-QROM.Verify $(\text{pkdS}, \text{pk}_\tau, \sigma_\tau)$ **then**
  return true
else
  return false
end if

4.3. Security Proof

Unforgeability. Since the proposed scheme is instantiated using the Kyber KEM and the Dilithium-QROM signature scheme, it inherits their established security guarantees. In particular, the unforgeability of the underlying primitives directly implies the following lemma.

Lemma 1 (Unforgeability)**.** *Let $\mathcal{A}$ be any probabilistic polynomial-time adversary against the unforgeability of the proposed DPS with security parameter λ, $p(\lambda)$ and $p'(\lambda)$ be polynomials in the security parameter λ. Then we have*

$$\mathbf{Adv}^{unforge}_{DPS,\mathcal{A}}(\lambda) \leq p(\lambda)\mathbf{Adv}^{IND\text{-}CCA}_{Kyber,\mathcal{B}}(\lambda) + p'(\lambda)\mathbf{Adv}^{SUF\text{-}CMA}_{Dilithium\text{-}QROM,\mathcal{B}}(\lambda).$$

Proof. We begin by observing that an adversary $\mathcal{A}$ wins the experiment $\mathrm{Exp}^{\mathrm{unforge}}_{DPS,\mathcal{A}}(\lambda)$ (Figure 3) if it succeeds in either breaking the delegation of warrants or producing a forged signature. This is reflected in Line 5 of the experiment, within the procedure Verify, where two verification checks are performed, namely, Dilithium-QROM.Verify$(\mathrm{pkdS}, \mathrm{pk}_\tau, \sigma_\tau)$ and Dilithium-QROM.Verify$(\mathrm{pk}_\tau, m, \sigma_m)$.

Accordingly, we decompose the advantage of $\mathcal{A}$ into two winning events, denoted by $\mathcal{E}_1$ and $\mathcal{E}_2$, where $\mathcal{E}_1$ corresponds to a forgery of σ_τ and $\mathcal{E}_2$ corresponds to a forgery of σ_m. We bound the adversary's advantage by the sum of the probabilities of these two events.

$$\Pr\left[\mathrm{Exp}^{\mathrm{unforge}}_{DPS,\mathcal{A}}(\lambda) = 1\right] = \Pr[\mathcal{E}_1 = 1] + \Pr[\mathcal{E}_2 = 1].$$

For the forgery of σ_τ, we argue directly that the probability that an adversary wins this game is bounded by the unforgeability of the digital signature scheme Dilithium-QROM, namely,

$$Pr[\mathcal{E}_1 = 1] \leq \mathbf{Adv}^{SUF\text{-}CMA}_{\text{Dilithium-QROM},\mathcal{B}}(\lambda).$$

For the forgery of σ_m, from Theorem 2 of [7] we have that

$$\Pr[\mathcal{E}_2 = 1] \leq p_1(\lambda) \cdot \mathbf{Adv}^{\text{SUF-CMA}}_{\text{Dilithium-QROM},\mathcal{B}}(\lambda) + p_2(\lambda) \cdot \mathbf{Adv}^{\text{KS}}_{\text{ARKG},\mathcal{B}}(\lambda),$$

where $p_1(\lambda)$ and $p_2(\lambda)$ are polynomials in the security parameter λ.

For the secret key security of our ARKG, in a secret key derivation procedure, we first invoke KyberKEM.Decap to recover the τ from the ciphertext. The value τ is then concatenated with t_1 and fed into a cryptographic hash function to generate pseudorandom key material. Therefore, the key security of our ARKG can be reduced to the IND-CCA security of Kyber, and we have

$$\mathbf{Adv}^{\text{KS}}_{\text{ARKG},\mathcal{B}}(\lambda) \leq \mathbf{Adv}^{\text{IND-CCA}}_{\text{Kyber},\mathcal{B}}(\lambda) + \varepsilon(\lambda),$$

where $\varepsilon(\lambda)$ is a polynomials that is negligible in λ. Let $p(\lambda) = p_2(\lambda)$ and $p'(\lambda) = 1 + p_1(\lambda)$, and finally we obtain

$$\mathbf{Adv}^{\text{unforge}}_{\text{DPS},\mathcal{A}}(\lambda) \leq p(\lambda)\mathbf{Adv}^{IND\text{-}CCA}_{\text{Kyber},\mathcal{B}}(\lambda) + p'(\lambda)\mathbf{Adv}^{SUF\text{-}CMA}_{\text{Dilithium-QROM},\mathcal{B}}(\lambda)$$

□

Distinguishability. When the verifier receives the signature of the message he requested with warr, double verification must be carried out. Namely, the verifier verifies the proxy certificate authorized by the delegator with the delegator's public key pkdS, and si-

multaneously the signature of the message he requests with the proxy public key pk_τ. After this procedure, the verifier can distinguish the proxy signatures and common signatures.

Verifiability. In the verification phase, the verifier first uses the delegator's long-term public key to validate the authorization credential warr generated for the proxy signer. Subsequently, the proxy public key contained in warr is used to verify the signature produced by the proxy signer. It is the first step that guarantees that the delegator has explicitly authorized the proxy to perform operations on its behalf.

Strong undeniability. The proxy key pair is jointly generated by the delegator and a designated proxy, ensuring uniqueness. Furthermore, due to the unforgeability of the scheme, no third party can generate a valid proxy signature σ_m. Once the signature successfully passes the verification of its authorization credential, the proxy signer cannot repudiate the signature generation action.

Unlinkability. This is mainly attributed to the indistinguishability during key derivation under the PSUW framework. The shared secret τ is computationally indistinguishable from a random string. Thus, $t'_{1,p}$ derived from τ is also random, leading to the distribution of pk_τ being computationally indistinguishable from a uniform sample over the public key space. Additionally, since the signature carries no trace of the secret key or the original identity, an adversary cannot identify the underlying signer from the signature, either.

4.4. Correctness Proof

Our construction consists of two main components: the ARKG scheme and Dilithium-QROM signature scheme. The correctness of the overall DPS scheme follows from the correctness guarantees of these two building blocks. We first formalize this intuition below and then prove each component separately.

Lemma 2 (Correctness of DPS)**.** *Let τ_1 and τ_2 denote the failure probabilities of ARKG and Dilithium-QROM, respectively. If both τ_1 and τ_2 are negligible in the security parameter λ, then the overall failure probability of the DPS scheme satisfies*

$$\tau \leq \tau_1 + \tau_2,$$

which is also negligible. Hence, the DPS scheme is correct with overwhelming probability.

Correctness of ARKG. Our ARKG construction is instantiated based on KyberKEM. We first recall the correctness notion of KyberKEM.

Definition 4 (KyberKEM Correctness [36])**.** *Let* $\mathrm{KyberKEM} = (\mathrm{KeyGen}, \mathrm{Encap}, \mathrm{Decap})$. *For* $(pk_{kem}, sk_{kem}) \leftarrow \mathrm{KeyGen}()$ *and* $(K, c) \leftarrow \mathrm{Encap}(pk_{kem})$*, the scheme is said to be δ_{kem}-correct if*

$$\Pr\left[\mathrm{Decap}(sk_{kem}, c) \neq K\right] \leq \delta_{kem}.$$

If $\delta_{kem} = 0$, the scheme is perfectly correct.

According to the Kyber specification [38], the decapsulation failure probability δ_{kem} is negligible across all security levels, ranging from 2^{-174} to 2^{-139}. To analyze the correctness of the ARKG, we adopt the notion introduced by Frymann et al. [6].

Definition 5 (ARKG Correctness)**.** *An ARKG scheme is correct if for all $\lambda \in \mathbb{N}$ and* $\mathrm{pp} \leftarrow \mathrm{Setup}(\lambda)$,

$$\Pr\left[\mathrm{ARKG.Check}(\mathrm{pk}_\tau, \mathrm{sk}_\tau) = \mathrm{true}\right] = 1,$$

where

$$\begin{aligned}(\mathrm{pk}_{kem}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS}) &\leftarrow \mathrm{ARKG.KeyGen}(\mathrm{pp});\\ (\mathrm{pk}_{\tau}, \mathrm{cred}) &\leftarrow \mathrm{ARKG.DerivePK}(\mathrm{pkpS}, \mathrm{pk}_{kem});\\ \mathrm{sk}_{\tau} &\leftarrow \mathrm{ARKG.DeriveSK}(\mathrm{cred}, \mathrm{sk}_{kem}, \mathrm{pkpS}, \mathrm{skpS}).\end{aligned}$$

We show in Appendix B that the derived key pair $(\mathrm{pk}_{\tau}, \mathrm{sk}_{\tau})$ always satisfies the ARKG.Check predicate, except with probability stemming from KEM decapsulation failure. Combining the above, the overall correctness error of ARKG is dominated by the KyberKEM failure probability, i.e.,

$$\tau_1 = \delta_{kem} \in [2^{-174}, 2^{-139}],$$

which is negligible.

Correctness of Dilithium-QROM. We next analyze the correctness of the signature component. For correctness, it is required that for any valid key pair $(\mathrm{pk}_{\tau}, \mathrm{sk}_{\tau})$ and any message m, a signature $\sigma \leftarrow \mathrm{Dilithium\text{-}QROM.Sign}(\mathrm{sk}_{\tau}, m)$ verifies successfully except with negligible probability.

Dilithium-QROM can be viewed as the Fiat–Shamir with aborts transform applied to an identification scheme [39], i.e., $\mathrm{FS}[ID, H, \kappa]$ [16], where κ denotes the maximum number of repetitions.

Lemma 3 (Lemma 4.4 of [16,31]). *Under the conditions*

$$\max_{s \in S_{\eta}, c \in ChSet} \|c\mathbf{s}_{1,\tau}\|_{\infty} \leq \beta, \quad \max_{t_0 \in S_{2^{d'}}, c \in ChSet} \|c\mathbf{t}_{0,\tau}\|_{\infty} \leq \gamma,$$

with $\beta \ll \gamma$, the identification scheme Dilithium-QROM-ID *has correctness error*

$$\delta_{ID} \approx 1 - \exp\left(-\beta n \left(\frac{k}{\gamma} + \frac{\ell}{\gamma}\right)\right).$$

Due to the use of rejection sampling (i.e., aborts), the signing algorithm repeats the identification protocol up to κ times until a valid transcript is obtained. Therefore, the overall failure probability of the signature scheme is

$$\tau_2 = \delta_{ID}^{\kappa}.$$

Following the parameterization in [14], we set $\kappa = \frac{200}{1-\delta_{ID}}$, which yields

$$\tau_2 \leq \exp(-200),$$

and hence τ_2 is negligible.

Combining the above results, the total failure probability of the DPS scheme satisfies

$$\tau \leq \tau_1 + \tau_2,$$

where both τ_1 and τ_2 are negligible. Therefore, the proposed DPS scheme is correct with overwhelming probability.

5. Performance Evaluation

In this section, we first compare the functionality of the proposed scheme with that of other proxy signature schemes [2,17–20,26,29]. Subsequently, we analyze the theoretical complexity of each post-quantum scheme and finally present our experimental results.

5.1. Functionalities

In terms of functionality, we compare our scheme with both post-quantum proxy signature schemes and classical proxy signature schemes. The comparison mainly focuses on two properties: asynchrony and unlinkability.

- Asynchrony: After the derived proxy public key is generated, the corresponding private key can be computed later by the legitimate key owner, thereby separating the generation process of the public–private key pair.
- QROM: QROM is a strict extension of the classical ROM into the quantum setting. While the classical ROM only models classical adversaries, QROM allows quantum adversaries to query the oracle in superposition [16]. It implies that QROM security is strictly stronger than ROM security and generally requires a separate proof.

Additionally, we stress that our scheme is secure under the QROM. This provides a tighter security guarantee compared with most existing post-quantum proxy signature schemes, whose security proofs are typically established in ROM. Table 2 shows the functionality comparison between our proposed DPS and other related solutions [2,17–20,26,29].

Table 2. Functionality comparison of different schemes.

Scheme	Post-Quantum Security	Unlinkability	Asynchrony	QROM
Li et al. [20]	✓	✓	×	×
Guo et al. [17]	✓	✓	×	×
Singh et al. [19]	✓	✓	×	×
Wang et al. [29]	✓	✓	×	×
Zhu et al. [18]	✓	✓	×	×
Verma et al. [2]	×	✓	×	×
Ghada et al. [26]	×	✓	✓	×
Ours	✓	✓	✓	✓

5.2. Theoretical Complexity

We consider theoretical complexity from the perspective of space complexity and theoretical time efficiency and compare our proposed DPS with the related post-quantum solutions [18–20,29].

Table 3 shows the key and signature size comparison between our proposed scheme and other lattice-based proxy signature schemes [17–19,29]. Here, q denotes the modulus and n denotes the dimension. And m represents the dimension of matrix $\mathbf{A}$, while k denotes the dimension of the secret vector. Our proxy signing key and signature structure are consistent with the module-based design of Dilithium. Benefiting from this structure, our scheme achieves significantly more compact sizes compared to traditional lattice-based ID-based proxy signature schemes.

Table 3. Key and signature size comparison of different schemes.

Scheme	PK	SK	Proxy-Signing Key	Signature
SRP+ [19]	$n \log q$	$2n \log(\sigma\sqrt{n})$	$2n \log(\sigma\sqrt{n})$	$6n \log q$
LSL+ [20]	$2mn \log q$	$2mkn \log q$	$3mkn \log q$	$9n \log q$
ZWW+ [18]	$nm \log q$	$mk \log(2d+1)$	$mk \log q$	$\lambda \log k + k \log q + m \log(12\sigma)$
WHC [29]	$mn \log q$	$m^2 \log q$	$m \log q$	$2m \log q$
Ours	$kn \log q$	$(2k+l)n \log(2\eta+1)$	$(2k+l)n \log(2\eta+1)$	$\ell n \log\big(2(\gamma-\beta)\big)$

Table 4 shows the theoretical time efficiency of each procedure in our signature scheme and comparisons with other schemes. We denote computational costs as t_h for the hash

algorithm, t_m for matrix multiplication, t_s for sample computation, and t_r for rejection sampling cost. t_{nk} denote the NIZK operation which is used in LSL+ [20]. t_b, t_e and t_p denote the computational costs of the BasisDel, ExtBasis and SamplePre algorithms, respectively, which are used in [17,18,29]. In the KeyGen and Proxy KeyGen phases, our scheme does not require the computationally intensive SamplePre and ExtBasis operation, resulting in significantly higher time efficiency compared to existing methods. During the Delegate and Proxy Sign phases, our scheme also avoids the costly SamplePre operation relative to LSL+ [20] and WHC [29], thereby achieving improved computational efficiency. Compared to SRP [19] and ZWW+ [18], our scheme requires fewer hash evaluations and matrix multiplications, offering a theoretical advantage in time efficiency. This improvement in efficiency is particularly beneficial in IoT scenarios, where computational resources are often limited.

Table 4. Theoretical time efficiency of different schemes.

Scheme	KeyGen	Delegate	Proxy KeyGen	Proxy Sign	Verify
SRP+ [19]	$t_h + t_s + t_p$	$t_h + 3t_m + 2t_s + t_r$	$t_h + t_s + t_p$	$5t_h + 6t_s + 9t_m + 2t_r$	$7t_h + 7t_m$
LSL+ [20]	$t_h + t_m + t_e$	$3t_h + 2t_m + t_e$	$t_h + t_m + t_e$	$3t_h + 2t_e + t_p + t_{nk}$	$t_h + t_{nk}$
ZWW+ [18]	$t_h + kt_p$	$t_h + t_s + 2t_m + t_r$	$2t_h + 2t_m + kt_p$	$3t_h + t_s + 3t_m + t_r$	$3t_h + 3t_m$
WHC [29]	$t_m + t_b$	$t_h + t_s + t_p$	$t_h + t_b + t_s$	$t_h + t_p$	$2t_h + 6t_m$
Ours	$2t_m + t_h + 4t_s$	$2t_h + 2t_m + 2t_s + t_r$	$t_m + t_h$	$t_h + t_s + 2t_m + t_r$	$2t_m + 2t_h$

5.3. Experimental Result

In this part, we present the implementation results with the related works [18–20,29] as detailed below. The performance is benchmarked across the five fundamental stages of a proxy signature scheme: KeyGen, Proxy KeyGen, Delegate, Proxy Sign, and Verify, where Proxy KeyGen corresponds to the DeriveSK algorithm in our scheme. All experiments run on the same machine with an AMD Ryzen 5 3600 6-Core @3.60 GHz and 32 GB RAM, running Ubuntu 22.04.6 LTS. Our code is publicly available at https://github.com/PQC-ProxySign/DPS (accessed on 10 May 2026). To ensure statistical reliability, all reported timing results are the average of 1000 independent executions. The recommended parameter sets of our DPS scheme are presented in Table 5 for the 128-bit security level in QROM. All performance data in Figure 5 were obtained under this unified setup, ensuring a fair comparison between our DPS scheme and the prior works.

Table 5. Recommended parameters for the DPS scheme.

Dilithium-QROM Parameters	
q	$2^{45} - 21{,}283$
n_s	512
(k, ℓ)	$(4, 4)$
d	15
$d' = \lfloor \frac{d}{2} \rfloor$	7
$\text{ChSet} = \{c \in R \mid \|c\|_\infty = 1 \wedge \|c\| = \dots\}$	$\sqrt{46}$
β	322
γ	905,679
η_s	7
BKZ-blocksize [13]	480
Kyber512 Parameters	
q_k	3329
k	2
BKZ-blocksize [16]	406

Remark 2. *We instantiate Kyber at NIST security level 1 (Kyber512) specified in [13] and adopt the Dilithium-QROM parameter set of [16]. As shown in Table 5, the two parameter sets yield BKZ block sizes of 406 [13] and 480 [16], respectively. Under the standard lattice attack cost model [33,35], both are estimated to provide no less than 128-bit security against lattice attacks. Thus, our scheme satisfies 128-bit security.*

As shown in Figure 5a, our scheme significantly outperforms SRP+ and ZWW+ in both the KeyGen and Proxy KeyGen phases. Specifically, the KeyGen time of our scheme is only 1.36 s, which is much lower than that of SRP+ [19] (15.75 s) and ZWW+ [18] (18.34 s). Similarly, in the Proxy KeyGen phase, our scheme requires only 1.24 s, while SRP+ [19] and ZWW+ [18] consume 16.65 s and 18.38 s, respectively. Overall, our scheme achieves approximately 13.5× speedup in the KeyGen phase and 14.8× speedup in the Proxy KeyGen phase compared with existing schemes, demonstrating its high efficiency in key generation-related operations.

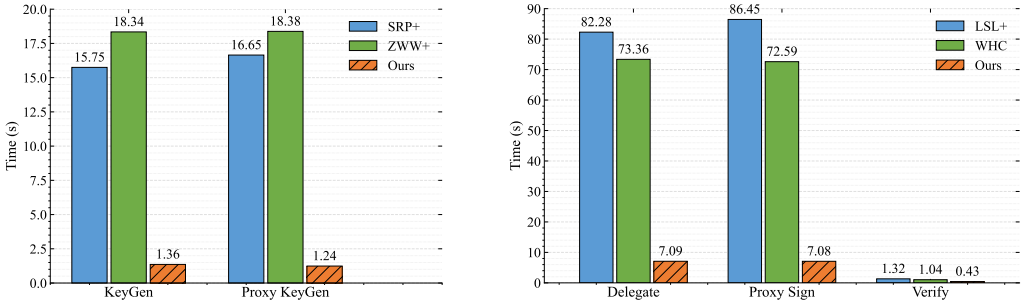

Figure 5. Performance comparison of different phases. (**a**) The comparison of Key Generation Performance. (**b**) The comparison of Delegate/Proxy Sign/Verify phase.

As shown in Figure 5b, our scheme exhibits significant performance advantages in the Delegate, Proxy Sign, and Verify phases. In the Delegate phase, our scheme completes the operation in only 7.09 s, achieving 11.6× and 10.3× speedups compared with LSL+ [20] and WHC [29], respectively. Similarly, in the Proxy Sign phase, our scheme only costs 7.08 s, which is 12.2× and 10.25× faster than LSL+ [20] and WHC [29]. In the verification phase, our scheme reduces the running time from 1.32 s and 1.04 s to 0.43 s, achieving approximately 3.1× and 2.4× speedups over LSL+ and WHC, respectively. Overall, our scheme achieves approximately a 10× speedup for both the delegation and proxy signing phases, as well as a 2.4× performance improvement in the verification phase, which further validates its high efficiency in practical scenarios.

6. Discussion

Choice of KEM. Our construction is instantiated with CRYSTALS-Kyber but is not restricted to it. Any post-quantum KEM satisfying CCA security can be used as the encapsulation primitive in DPS.ARKG. Kyber is chosen primarily for its natural compatibility with Dilithium: both are NIST-standardized, MLWE-based, and supported by QROM security proofs. This yields a uniform post-quantum security foundation for the overall DPS scheme. Owing to the modularity of ARKG, alternative post-quantum KEMs can be incorporated without severely changing the high-level proxy-signature design.

Implementation and Performance.We provide both theoretical and empirical evaluations of our lattice-based proxy signature scheme. And the experimental results show computational gains over existing identity-based post-quantum schemes. Notably, most

lattice-based proxy constructions lack concrete benchmarks, highlighting opportunities for future work to optimize and rigorously compare these schemes.

Trade-offs and Limitations. As inherent to lattice-based cryptography, our scheme incurs larger public key and signature sizes and longer execution times than classical mathematical problem-based counterparts. While conventional proxy signatures typically run in milliseconds, lattice-based constructions execute in seconds. Integrating lattice-based schemes with specialized hardware is a key avenue for enabling deployment in resource-constrained environments.

Broader Applicability. Beyond the IoT, the scheme's core framework supports post-quantum secure delegation in diverse authorization scenarios, providing a foundation for broader applications and future research.

7. Conclusions

In this work, we first propose an ARKG scheme based on Kyber to support asynchronous key generation for Dilithium-QROM. Building upon this, we instantiate the first Dilithium-based proxy signature (DPS) under the PSUW framework, which eliminates the need for lattice trapdoors and thus achieves higher efficiency. Furthermore, our scheme supports asynchronous proxy key generation and unlinkability among derived keys, thereby enhancing the flexibility and practical applicability of the scheme in delegated IoT scenarios. We implement our scheme and evaluate it against the state-of-the-art approaches. Experimental results demonstrate that our scheme achieves significant performance improvements in the delegation, proxy signing, and verification phases.

Author Contributions: Conceptualization, Y.W., R.D. and T.Y.; methodology, Y.W., R.D. and T.Y.; software, Z.H. and Y.W.; validation, Z.H.; formal analysis, Y.W., R.D. and T.Y.; data curation, Z.H.; writing—original draft preparation, Y.W., Z.H., R.D. and T.Y.; writing—review and editing, R.D., T.Y., Y.W. and Z.H.; supervision, J.W. (Jian Weng) and J.W. (Jiasi Weng); project administration, J.W. (Jian Weng); funding acquisition, J.W. (Jian Weng). All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (Nos. 62332007, U22B2028, 62302192); the Shenzhen Major Science and Technology Special Project (No. KJZD20240903101108011); the Guangdong Special Support Program for Distinguished Talents (No. 2024JC08X015); the Joint Project of the National Natural Science Foundation of China (No. U23A20303); the General Project of the Guangdong Provincial Natural Science Foundation (Nos. 2024A1515010086, 2026A1515010190).

Data Availability Statement: The original contributions presented in this study are included in the article; further inquiries or requests can be directed to the corresponding authors.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Oracles for the Unforgeability Experiment

The adversary $\mathcal{A}$ is allowed to make a polynomial number of queries to the following oracles in the unforgeability experiment.

- $\mathcal{O}_{\text{Reg}}$: Upon invocation, this oracle samples a fresh key pair $(\text{skpS}, \text{pkpS})$, stores it in the list $\mathcal{L}_{\text{Reg}}$, and returns pkpS.
- $\mathcal{O}_{\text{Delegate}}(\text{pkpS})$: This oracle is initialized with the delegator's private key skdS, and proxy's encapsulation public key pk_{kem}. Upon receiving a proxy's signature public key pkpS as input, it computes $\text{Delegate}(\text{pkpS}, \text{pk}_{kem}, \text{skdS})$, records the output in the list $\mathcal{L}_{\text{Del}}$, and returns warr and ddata to the adversary $\mathcal{A}$, without revealing skdS. If $(\cdot, \text{pkpS}) \notin \mathcal{L}_{\text{Reg}}$, the oracle aborts. This oracle models an honest delegation to pkpS.

- $\mathcal{O}_{\text{ProxySign}}(\text{pkpS}, \text{warr}, \text{ddata}, m)$: This oracle is initialized with the delegator's public key pkdS, the proxy's decapsulation private key sk_{kem}. Upon receiving as input a proxy's signature public key pkpS, a warrant warr, delegation data ddata, and a message m, it retrieves the corresponding secret key skpS from $\mathcal{L}_{\text{Reg}}$ such that $(\text{skpS}, \text{pkpS}) \in \mathcal{L}_{\text{Reg}}$, computes ProxySign(pkpS, skpS, warr, ddata, sk_{kem}, m), and records the resulting signature in the list $\mathcal{L}_{\text{Sign}}$. If ProxySign outputs $\perp$, the oracle aborts. This oracle models a signing query on a message chosen by the adversary $\mathcal{A}$.
- $\mathcal{O}_{\text{Corr}}(\text{pkpS})$: Upon receiving a proxy's signature public key pkpS as input, this oracle retrieves the corresponding secret key skpS from the list $\mathcal{L}_{\text{Reg}}$, returns it to the adversary, and records skpS in the list $\mathcal{L}_{\text{Corrupt}}$. If $(\cdot, \text{pkpS}) \notin \mathcal{L}_{\text{Reg}}$, the oracle aborts. This oracle models the leakage of the proxy's signature secret key.

Appendix B. Proof of ARKG.Check

Proof. To prove $\Pr\left[\text{ARKG.Check}(\text{pk}', \text{sk}') = \text{true}\right] = 1$ for simplicity, we substitute each coefficient on $\mathbf{t}_\tau$ with T, $\mathbf{t}_p, \mathbf{t}'_p$ with T_1, T_2, and $\mathbf{t}_{1,\tau}, \bar{\mathbf{t}}_{1,p}, \mathbf{t}'_{1,p}, \bar{\mathbf{t}}_{1,\tau}$ with t, t'_1, t'_2, t'. According to Algorithms 1–3, we have $T, T_1, T_2 \in \mathbb{Z}_q[X]$ satisfy the following:

$$\begin{cases} T &= T_1 + T_2 \\ t &= \text{Power2Round}_q(T, d') \\ t'_1 &= \text{Power2Round}_q(T_1, d' - 1) \\ t'_2 &= \text{Power2Round}_q(T_2, d' - 1) \\ t' &= t'_1 + t'_2 \end{cases}$$

By the definition of Power2Round, there exists t_0 such that $T = t2^{d'} + t_0$, where $t_0 \in (-2^{d'-1}, 2^{d'-1}]$. Similarly, r_1, r_2 satisfy $T_1 = t'_1 2^{d'-1} + r_1$, $T_2 = t'_2 2^{d'-1} + r_2$ with $r_1, r_2 \in (-2^{d'-2}, 2^{d'-2}]$. Note that T also equals the sum of $(t'_1 2^{d'-1} + r_1)$ and $(t'_2 2^{d'-1} + r_2)$. Hence, we have

$$(t'_1 + t'_2)2^{d'-1} + r_1 + r_2 = t2^{d'} + t_0, \quad t'2^{d'-1} = t2^{d'} + t_0 - (r_1 + r_2).$$

From the bounds on t_0, r_1, r_2, we obtain $\Delta = t_0 - (r_1 + r_2) \in (-2^{d'}, 2^{d'}]$.
When it comes to calculating $\|\alpha_1 - \alpha_2\|_\infty$ in Algorithm 4, where

$$\alpha_1 = \text{Power2Round}_q(t2^{d'}, d') = t$$
$$\alpha_2 = \text{Power2Round}_q(t'2^{d'-1}, d') = \mathit{Power2Round}_q(t2^{d'} + \Delta, d'),$$

with the bound of Δ, the result is influenced by the carry generated from the lower position. Therefore $|\alpha_1 - \alpha_2| \leq 1$. Applying this bound coefficient-wise yields $\|\alpha_1 - \alpha_2\|_\infty \leq 1$. □

References

1. Mansoor, K.; Afzal, M.; Iqbal, W.; Abbas, Y. Securing the future: Exploring post-quantum cryptography for authentication and user privacy in IoT devices. *Clust. Comput.* **2025**, *28*, 93. [CrossRef]
2. Verma, G.K.; Singh, B.; Kumar, N.; Obaidat, M.S.; He, D.; Singh, H. An efficient and provable certificate-based proxy signature scheme for IIoT environment. *Inf. Sci.* **2020**, *518*, 142–156. [CrossRef]
3. Mambo, M.; Usuda, K.; Okamoto, E. Proxy signatures for delegating signing operation. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*; Association for Computing Machinery: New York, NY, USA, 1996; pp. 48–57.
4. Boldyreva, A.; Palacio, A.; Warinschi, B. Secure proxy signature schemes for delegation of signing rights. *J. Cryptol.* **2012**, *25*, 57–115. [CrossRef]
5. Frymann, N.; Gardham, D.; Manulis, M. Asynchronous remote key generation for post-quantum cryptosystems from lattices. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*; IEEE: Piscataway, NJ, USA, 2023; pp. 928–941.

6. Frymann, N. Asynchronous Remote Key Generation and its Applications in W3C WebAuthn and FIDO. Ph.D. Thesis, University of Surrey, Surrey, UK, 2024.
7. Frymann, N.; Gardham, D.; Manulis, M. Unlinkable delegation of webauthn credentials. In *Proceedings of the European Symposium on Research in Computer Security*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 125–144.
8. Bannore, A.; Patil, R.Y.; H. Patil, Y.; Deshpande, H. Proxy signature-based role delegation scheme: Formal analysis and simulation. *Int. J. Inf. Technol.* **2024**, *16*, 4027–4038. [CrossRef]
9. Hussain, S.; Tufail, A.; Naim, H.A.A.G.; Khan, M.A.; Barb, G. A Lightweight Proxy Signature Scheme for Resource-Constrained NDN-based Internet of Vehicles. *IEEE Open J. Veh. Technol.* **2025**, *6*, 2607–2626. [CrossRef]
10. Hundera, N.W.; Mei, Q.; Xiong, H.; Geressu, D.M. A Secure and Efficient Identity-Based Proxy Signcryption in Cloud Data Sharing. *KSII Trans. Internet Inf. Syst.* **2020**, *14*, 455–472. [CrossRef]
11. Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehle, D. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *IEEE European Symposium on Security and Privacy (EuroS&P)*; IEEE: Piscataway, NJ, USA, 2018; pp. 353–367.
12. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2018**, *2018*, 238–268. [CrossRef]
13. National Institute of Standards and Technology (NIST). Module-Lattice-Based Key-Encapsulation Mechanism Standard, 2024. Available online: https://csrc.nist.gov/pubs/fips/203/final (accessed on 13 August 2024).
14. National Institute of Standards and Technology (NIST). Module-Lattice-Based Digital Signature Standard, 2024. Available online: https://csrc.nist.gov/pubs/fips/204/final (accessed on 10 May 2026).
15. Hanna, Y.; Bozhko, J.; Tonyali, S.; Harrilal-Parchment, R.; Cebe, M.; Akkaya, K. A comprehensive and realistic performance evaluation of post-quantum security for consumer IoT devices. *Internet Things* **2025**, *33*, 101650. [CrossRef]
16. Kiltz, E.; Lyubashevsky, V.; Schaffner, C. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 552–586.
17. Guo, H.; Tian, K.; Liu, F.; Zheng, Z.; Wang, Y. Identity-based linearly homomorphic proxy signature scheme. *Comput. Netw.* **2025**, *272*, 111703. [CrossRef]
18. Zhu, H.; Wang, Y.; Wang, C.; Cheng, X. An efficient identity-based proxy signcryption using lattice. *Future Gener. Comput. Syst.* **2021**, *117*, 321–327. [CrossRef]
19. Singh, S.; Rawal, S.; Padhye, S.; Tiwari, N. Identity based proxy blind signature scheme using NTRU lattices. *Inf. Comput.* **2025**, *304*, 105284. [CrossRef]
20. Li, Q.; Shen, J.; Lin, C.; Wang, Z.; He, D. Two-Round Identity-Based Proxy Blind Signature Scheme on Lattices. *IEEE Internet Things J.* **2025**, *12*, 33621–33634. [CrossRef]
21. Jia, X.; He, D.; Liu, Q.; Choo, K.K.R. An efficient provably-secure certificateless signature scheme for Internet-of-Things deployment. *Ad Hoc Netw.* **2018**, *71*, 78–87. [CrossRef]
22. Karati, A.; Islam, S.H.; Karuppiah, M. Provably secure and lightweight certificateless signature scheme for IIoT environments. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3701–3711. [CrossRef]
23. Singh, H.; Verma, G.K. ID-based proxy signature scheme with message recovery. *J. Syst. Softw.* **2012**, *85*, 209–214. [CrossRef]
24. James, S.; Thumbur, G.; Reddy, P.V. Secure and efficient identity-based proxy signature scheme with message recovery. *J. Math. Comput. Sci.* **2020**, *10*, 448–473. [CrossRef]
25. Liu, Y.; Wang, L.; Chen, H.H. Message authentication using proxy vehicles in vehicular ad hoc networks. *IEEE Trans. Veh. Technol.* **2014**, *64*, 3697–3710. [CrossRef]
26. Almashaqbeh, G.; Nitulescu, A. Anonymous, timed and revocable proxy signatures. In *Proceedings of the International Conference on Information Security*; Springer: Berlin/Heidelberg, Germany, 2024; pp. 23–43.
27. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*; IEEE: Piscataway, NJ, USA, 1994; pp. 124–134.
28. Proos, J.; Zalka, C. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv* **2003**, arXiv:quant-ph/0301141. [CrossRef]
29. Wang, L.; Huang, C.; Cheng, H. Novel proxy signature from lattice for the post-quantum Internet of Things. *J. Ambient. Intell. Humaniz. Comput.* **2023**, *14*, 9939–9946. [CrossRef]
30. Cash, D.; Hofheinz, D.; Kiltz, E.; Peikert, C. Bonsai trees, or how to delegate a lattice basis. *J. Cryptol.* **2012**, *25*, 601–639. [CrossRef]
31. Eaton, E.; Stebila, D.; Stracovsky, R. Post-quantum key-blinding for authentication in anonymity networks. In *Proceedings of the International Conference on Cryptology and Information Security in Latin America*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 67–87.
32. Fujisaki, E.; Okamoto, T. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *Proceedings of the Advances in Cryptology—CRYPTO' 99*; Wiener, M., Ed.; Springer: Berlin/Heidelberg, Germany, 1999; pp. 537–554.

33. Alkim, E.; Ducas, L.; Pöppelmann, T.; Schwabe, P. Post-quantum key exchange: A new hope. In Proceedings of the 25th USENIX Conference on Security Symposium, Austin, TX, USA, 10–12 August 2016; pp. 327–343.
34. Albrecht, M.R.; Player, R.; Scott, S. On the Concrete Hardness of Learning with Errors. *Cryptology ePrint Archive*, 2015. Available online: https://eprint.iacr.org/2015/46 (accessed on 10 May 2026).
35. Hu, X.; Cao, Y.; Xiang, H. A review of lattice cryptography attack cost model. In *Applied Cryptography and Network Security Workshops*; Springer: Cham, Switzerland, 2026; pp. 41–60.
36. Maram, V.; Xagawa, K. Post-quantum anonymity of Kyber. In *Proceedings of the IACR International Conference on Public-Key Cryptography*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 3–35.
37. Lee, B.; Kim, H.; Kim, K. Secure mobile agent using strong non-designated proxy signature. In *Proceedings of the Australasian Conference on Information Security and Privacy*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 474–486.
38. Avanzi, R.; Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D.; et al. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round* **2019**, *2*, 1–43.
39. Lyubashevsky, V. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 598–616.