Article

Seamless Transition to Post-Quantum TLS 1.3: A Hybrid Approach Using Identity-Based Encryption

Thiago Leucz Astrizi * and Ricardo Felipe Custódio

Posted Date: 30 August 2024

doi: 10.20944/preprints202408.2219.v1

Keywords: n/a; Hybrid Post-Quantum Cryptography; KEMTLS; Network Security

Article

Seamless Transition to Post-Quantum TLS 1.3: A Hybrid Approach Using Identity-Based Encryption

Thiago Leucz Astrizi [1,*] and Ricardo Felipe Custódio [1,2]

Graduate Program on Computer Science, Department of Informatics and Statistics, Federal University of Santa Catarina (UFSC), Florianópolis 88040-370, SC, Brazil
* Correspondence: thiago.leucz.astrizi@posgrad.ufsc.br

Abstract: This paper proposes a novel approach to transitioning the classical TLS 1.3 protocol to a post-quantum version with minimal changes to the existing infrastructure. By integrating a hybrid KEMTLS protocol with Identity-Based Encryption (IBE), the solution achieves post-quantum security without incorporating post-quantum keys into TLS certificates. This enables the protocol to function with Certificate Authorities that do not support post-quantum keys, ensuring its security while simplifying the transition. The main challenge addressed is the immediate need for support of post-quantum algorithms by Certificate Authorities and servers. The proposed solution leverages existing certificates and keys, facilitating a smoother transition.

Keywords: hybrid post-quantum cryptography; KEMTLS; network security

1. Introduction

Transport Layer Security (TLS) is a widely used cryptographic protocol ensuring secure communication over computer networks. It is crucial for safeguarding web pages and machine-to-machine interactions. TLS 1.3, outlined in RFC 8446 [1], facilitates secure message exchange across untrusted networks. This is achieved by establishing a shared secret between communicating parties. Unlike purely symmetric encryption, TLS combines:

Diffie-Hellman Key Agreement Protocol:

This protocol enables the client and server to agree on a shared secret key during a handshake process without directly exchanging it. This is achieved through asymmetric cryptography, allowing both parties to compute the shared secret from public components independently;

Digital Signatures:

TLS employs digital signatures to authenticate the server and, optionally, the client. This process relies on public-key cryptography, utilizing public and private key pairs. Certificate Authorities (CAs) issue digital certificates to verify the authenticity of these public keys, ensuring trust in the communication parties.

Once established, the shared secret enables TLS to employ symmetric encryption for efficient bulk data transfer, safeguarding confidentiality and integrity. This combined use of asymmetric and symmetric cryptography forms a robust and efficient security foundation for TLS.

The emergence of Cryptographically Relevant Quantum Computers (CRQCs) poses a substantial threat to the security of TLS 1.3. A CRQC possesses sufficient computational power and stability to execute Shor's algorithm efficiently, enabling it to factor large integers and solve discrete logarithms exponentially faster than classical computers [2]. This capability undermines the security of current public-key cryptographic algorithms, including RSA and Elliptic Curve Cryptography (ECC). To counter this impending threat, the National Institute of Standards and Technology (NIST) launched a post-quantum cryptography standardization program in 2016 [3]. By 2022, NIST had already selected several post-quantum algorithms for subsequent testing and implementation, laying the groundwork for secure communication in the post-quantum era.

Adapting TLS 1.3 to resist quantum attacks typically involves replacing the Diffie-Hellman key agreement with a post-quantum key encapsulation mechanism (KEM) and substituting classical signatures with post-quantum ones. These modifications often increase message sizes due to the larger post-quantum keys and signatures. One approach to mitigate this issue is the Key Encapsulation Mechanism for Transport Layer Security (KEMTLS) protocol, developed by Peter Schwabe, Douglas Stebila, and Thom Wiggers [4]. KEMTLS employs post-quantum KEMs for both key exchange and authentication, resulting in more compact message sizes compared to traditional TLS handshakes. However, this approach introduces an additional round trip for secret sharing, potentially impacting latency.

The transition to post-quantum technology is anticipated to be gradual, with hybrid approaches combining classical and post-quantum algorithms to achieve security levels equivalent to the strongest component. This strategy mitigates risks associated with the nascent nature of post-quantum algorithms, which require rigorous validation. The recent compromise of SIKE, despite its advancement in the NIST standardization process, underscores the importance of cautious adoption and continuous evaluation [5,6].

The TLS ecosystem encompasses more than just the protocol itself; it includes certification authorities (CAs), certificate validation software, revocation status checking, trust in root CAs, and updates to client and server applications. Transitioning to post-quantum technologies necessitates significant updates across this ecosystem, including modifications to CAs, OCSP responders, and Certificate Revocation Lists (CRLs). Several factors may impede this transition. CAs must support both classical and post-quantum algorithms to issue and validate certificates during this transitional period. Moreover, the process of generating new keys and certificates for millions of TLS servers is complex and time-consuming. While large organizations may manage this efficiently, smaller entities and IoT devices with limited resources will likely face considerable challenges and delays.

To address these challenges, we propose a solution that enables post-quantum security for TLS communications without mandating immediate support for post-quantum algorithms from CAs or servers. This approach allows systems to adopt post-quantum TLS while retaining existing certificates and keys, streamlining the transition process. Compared to other proposals, such as KEMTLS, our solution minimizes ecosystem modifications, facilitating broader and faster adoption.

Our Proposal:

We have developed a hybrid KEMTLS variant combining post-quantum and classical key encapsulation mechanisms. The classical component leverages the existing public key infrastructure, while the post-quantum component utilizes identity-based encryption (IBE), eliminating the need for post-quantum keys within TLS certificates. This enables the use of existing certificates for post-quantum communications, preserving current certificate generation and authentication methods. Furthermore, current ECDSA-based leaf certificates can be employed for both message signing in TLS 1.3 and message encapsulation within our proposed protocol. We demonstrate that using the same key for both ECDSA signatures and a specific Diffie-Hellman-based KEM does not compromise security, following established ECDSA security models.

Identity-based systems employ a server's domain name or another identifier as its public key. This eliminates the need to transmit or certify public keys, as they are inherently public knowledge. However, secret keys for individual identities are generated by a Private Key Generator (PKG) and securely delivered to the corresponding servers.

Given our approach, one might question the necessity of a traditional public key infrastructure (PKI). While a fully hybrid scheme utilizing identity-based encryption (IBE) for both components is theoretically possible, eliminating the need for certificates and certificate authorities, practical challenges arise for general TLS usage. Specifically, establishing agreement on the Private Key Generator (PKG) during initial communication presents difficulties. Existing proposals for IBE-based post-quantum TLS often address this by focusing on constrained environments with known PKGs, such as IoT devices (e.g., Scott, 2023 [7] and Ducas, 2014 [8]). Our proposal aims for broader applicability by incorporating PKG information within TLS extensions, either in certificates or the protocol itself, while leveraging existing infrastructure for the classical component of the hybrid scheme.

1.1. Addressing IBE Challenges in TLS

Our proposal addresses two key challenges of integrating identity-based encryption (IBE) into TLS.

Revocation Problem: Employing a server's domain name as its identity presents a revocation challenge. Revoking the server's secret key would inadvertently revoke the entire domain, which is impractical. Common solutions involve appending timestamps or additional strings to the identity, often requiring additional certification. Our proposal addresses this by incorporating the classical public key from the certificate into the identity, aligning with existing revocation mechanisms.

Key Escrow Problem: IBE's reliance on a Private Key Generator (PKG) to produce secret keys for all parties poses a key escrow risk in a widely deployed protocol like TLS. Our hybrid approach addresses this by combining IBE with a traditional public key algorithm. Employing combiners, we ensure the scheme's security is at least as robust as its strongest component. A compromised PKG does not necessarily compromise the entire system, as the key encapsulation mechanism remains secure as long as the classical public key algorithm is unbroken. In the worst-case scenario of a leaked server key, the protocol degrades to a classical Diffie-Hellman-based system, maintaining TLS 1.3 security levels but losing post-quantum resistance. This trade-off is considered acceptable for a transitional solution facilitating migration to full post-quantum security. As the ecosystem matures, the IBE component can be replaced by more traditional post-quantum PKI mechanisms.

1.2. Paper Content

This paper is organized as follows.

Section 2 introduces the notation used throughout the paper and describes the cryptographic algorithms central to our proposal and the KEMTLS protocol. It includes a detailed breakdown of key encapsulation mechanisms (KEMs), identity-based encryption (IBE), and their integration within the TLS protocol.

Section 3 presents our novel integration of post-quantum identity-based encryption algorithms with classical public key algorithms. We develop a security model for the resulting hybrid cryptographic primitive and discuss its practical and theoretical security implications.

Section 4 compares the performance of our hybrid KEMTLS protocol with the current TLS 1.3 implementation and other hybrid approaches. Performance metrics are evaluated in both simulated network environments and real-world Internet configurations.

Section 5 concludes by summarizing our findings and discussing potential directions for future research in post-quantum cryptography within network security protocols. This section reflects on the broader implications for cybersecurity and the transition to quantum-resistant cryptographic systems.

Appendix A contains the full security proof supporting the claims made about the hybrid KEMTLS's resilience against classical and quantum cryptographic attacks as outlined in Theorem 1.

Appendix B provides the complete security proof of Theorem 2, substantiating the claims that we can use elliptic curve keys alternately to instantiate our proposed KEM algorithm and ECDSA signatures algorithms without compromising the security of either.

2. Preliminaries

2.1. Key Enchapsulation Mechanisms (KEMs)

The following algorithms define a Key Encapsulation Mechanism (KEM):

- **Key Generation:** $KEM.KeyGen(1^\lambda) \rightarrow (pk, sk)$: Given the security parameter 1^λ, this probabilistic polynomially-bounded algorithm generates a secret key sk and a public key pk.
- **Encapsulation:** $KEM.Encaps(pk) \rightarrow (k, c)$: Given a public key pk, this probabilistic polynomially-bounded algorithm generates a secret $k \in K$ and a ciphertext $c \in C$, where K is the key space and C the ciphertext space.
- **Decapsulation:** $KEM.Decaps(sk, c) \rightarrow k$: Given a secret key sk and a ciphertext $c \in C$, this deterministic polynomially-bounded algorithm recovers the corresponding secret $k \in K$.

The security of a public-key KEM is evaluated using two attack games (Game 0 and Game 1), as detailed bellow in KEM Attack Game 0 and KEM Attack Game 1. For any adversary $\mathcal{A}$, let W_0 denote the event where $\mathcal{A}$ returns 1 in Attack Game 0 and W_1 denotes the event where $\mathcal{A}$ returns 1 in Attack Game 1. The advantage of adversary $\mathcal{A}$ is defined as $Adv_{KEM}^{IND-CCA}(\mathcal{A}) = |Pr[W_0] - Pr[W_1]|$. A KEM is secure if the advantage is a negligible value for all adversaries $\mathcal{A}$.

KEM Attack Game 0

```
1: $(sk, pk) \leftarrow \texttt{KEM.KeyGen}(1^\lambda)$
2: $(c^*, k_0) \leftarrow \texttt{KEM.Encaps}(pk)$
3: $b \leftarrow \mathcal{A}^{\texttt{KEM.Decaps}(sk,\cdot)}(pk, c^*, k_0)$      ▷ Run adversary. It can perform decapsulation queries.
4:                                      ▷ In such queries, the adversary control the ciphertext to be decapsulated.
5:                                      ▷ But the secret key $sk$ is kept secret and cannot be interacted with.
6: return $b$
```

KEM Attack Game 1

```
1: $(sk, pk) \leftarrow \texttt{KEM.KeyGen}(1^\lambda)$
2: $(c^*, -) \leftarrow \texttt{KEM.Encaps}(pk)$
3: $k_1 \xleftarrow{\$} K$                                   ▷ Randomly chosen key from key space $K$
4: $b \leftarrow \mathcal{A}^{\texttt{KEM.Decaps}(sk,\cdot)}(pk, c^*, k_1)$
5: return $b$
```

KEM Attack Games:

The adversary's goal is to distinguish between two scenarios. The advantage measures the adversary's ability to distinguish between the games, compromising the KEM's security under chosen-ciphertext attacks.

In both games, the adversary can query an oracle to decapsulate messages using the secret key sk, but cannot cannot request the decapsulation of c^*.

When KEMs are used for key exchange, users typically generate their key pair independently of their identity. A certificate signed by a trusted authority, containing both pk and the identity of its owner, is used to verify that a claimed identity matches a present public key.

2.1.1. Diffie-Hellman KEM

A typical Key Encapsulation Mechanism (KEM) is the Diffie-Hellman KEM (DHKEM), defined over a multiplicative group G with generator g and order q, derived from an elliptic curve. The security of DHKEM relies on the computational difficulty of the Diffie-Hellman problem: given g, g^a and g^b, compute g^{ab}. The KEM employs a hash function $H : \{0,1\}^* \rightarrow K$ where K is the secret space and assumes that there are efficient algorithms to verify solutions for Diffie-Hellman problem.

The secret key is a random exponent x such that $0 < x < q$, and the corresponding public key is $y = g^x$. The encapsulation and decapsulation algorithms are detailed in Algorithms 3 and 4.

This KEM is described in more details and has a proposed standardization in Section 3.1 of RFC 9180 [9].

Algorithm 3 DHKEM.Encaps(y)

```
1: Input: Public key $y$
2: Output: Ciphertext $c$, Shared secret $k$
3: $x' \xleftarrow{\$} \mathbb{Z}_q$
4: $y' \leftarrow g^{x'}$
5: $s \leftarrow y^{x'}$
6: $k \leftarrow H(s \parallel y' \parallel y)$
7: $c \leftarrow y'$
8: return $(c, k)$
```

Algorithm 4 DHKEM.Decaps(x, c)

```
1: Input: Secret key $x$, Ciphertext $c$
2: Output: Shared secret $k$
3: $y' \leftarrow c$
4: $s \leftarrow y'^x$
5: return $H(s \parallel y' \parallel y)$
```

2.2. Identity Based KEMs

An identity-based Key Encapsulation Mechanism (IDKEM) comprises of the following algorithms:

- **Key Generation:** $IDKEM.KeyGen(1^\lambda) \rightarrow (mpk, msk)$: Given the security parameter 1^λ, this probabilistic polynomially-bounded algorithm generates a master public key mpk and a master secret key msk;
- **Key Extraction:** $IDKEM.Extract(msk, ID) \rightarrow sk_{ID}$: Given the master secret key msk and a string ID, this probabilistic polynomially-bounded algorithm generates a secret key sk_{ID} associated with identity ID;
- **Encapsulation:** $IDKEM.Encaps(mpk, ID) \rightarrow (k, c)$: Given the master public key mpk and an identity string ID, this probabilistic polynomially-bounded algorithm produces a secret k, and a ciphertext c;
- **Decapsulation:** $IDKEM.Decaps(sk_{ID}, c) \rightarrow k$: Given a secret key sk_{ID} associated with identity ID and a ciphertext c, this deterministic polynomially-bounded algorithm recovers the secret k.

The security of an ID-KEM is evaluated through attack games, which are analogous to those used for evaluating traditional KEMs. These games are depicted in ID-KEM Attack Game 0 and 1, and in Algorithm 7.

ID-KEM Attack Game 0

```
1: $(msk, mpk) \leftarrow \texttt{IDKEM.KeyGen}(1^\lambda)$
2: $(ID^*, st) \leftarrow \mathcal{A}_1^{\texttt{Decaps}(msk,\cdot),\texttt{IDKEM.Extract}(msk,\cdot)}(mpk)$
3: $(c^*, k_0) \leftarrow \texttt{IDKEM.Encaps}(mpk, ID^*)$
4: $b \leftarrow \mathcal{A}_2^{\texttt{Decaps}(msk,\cdot),\texttt{IDKEM.Extract}(msk,\cdot)}(st, c^*, k_0)$      ▷ Queries to Decaps are defined in Algorithm 7.
5: return $b$
```

The adversary's objective is to distinguish between the two scenarios presented in the attack games. The adversary's advantage, denoted as $Adv_{IDKEM}^{IND-CCA}(\mathcal{A})$, measures their ability to differentiate between the games. This is defined as $|\Pr[W_0] - \Pr[W_1]|$, where W_0 and W_1 represent the events that the adversary outputs 1 in Attack Game 0 and Attack Game 1, respectively. A secure ID-KEM should ensure that this advantage is negligible for any adversary $\mathcal{A}$.

ID-KEM Attack Game 1

```
1: $(msk, mpk) \leftarrow \texttt{IDKEM.KeyGen}(1^\lambda)$
2: $(ID^*, st) \leftarrow \mathcal{A}_1^{\texttt{Decaps}(msk,\cdot),\texttt{IDKEM.Extract}(msk,\cdot)}(mpk)$
3: $(c^*, -) \leftarrow \texttt{IDKEM.Encaps}(mpk, ID^*)$
4: $k_1 \xleftarrow{\$} K$                                  ▷ Randomly chosen key from key space $K$
5: $b \leftarrow \mathcal{A}_2^{\texttt{Decaps}(msk,\cdot),\texttt{IDKEM.Extract}(msk,\cdot)}(st, c^*, k_1)$
6: return $b$
```

Algorithm 7 Decapsulation Query for ID-KEM Attack Games

```
1: Decaps$(msk, ID, c)$:
2:     $sk_{ID} \leftarrow \texttt{IDKEM.Extract}(msk, ID)$
3:     return $\texttt{IDKEM.Decaps}(sk_{ID}, c)$
```

In the context of identity-based KEM attack games, the adversary can choose the identity it wishes to target. Initially, the adversary is given the master public key mpk and can send decapsulation queries to an oracle as defined in Algorithm 7. Additionally, the adversary can query the oracle for the output of IDKEM.Extract to obtain the secret key for any identity. After gathering this information, the adversary selects its target identity ID^*. The adversary then receives a ciphertext associated with the chosen identity and either the secret encapsulated in the ciphertext or an unrelated secret chosen uniformly at random. The adversary's objective is to distinguish between these two scenarios.

2.3. Message Authentication Codes (MACs)

A Message Authentication Code (MAC) ensures the integrity and authenticity of a message. Defined over the sets (K, M, T), a MAC comprises the following algorithms:

- **Key Generation:** $MAC.KeyGen(1^\lambda) \rightarrow key$: Given the security parameter 1^λ, this algorithm generate a key $key \in K$.
- **Signing:** $MAC.Sign(key, msg) \rightarrow tag$: Given inputs $key \in K$ and $msg \in M$, this algorithm generates a tag $tag \in T$.
- **Verifying signature:** $MAC.Verify(key, msg, tag) \rightarrow b$: Given a key $key \in K$, a message $msg \in M$, and a tag $tag \in T$, this algorithm outputs a value b, indicating either `accept` or `reject`.

A tag generated by $MAC.Sign$ for a message is always accepted when verified with the same message and key.

For our hybrid construction, we assume that a MAC is secure if no adversary can forge an accepted tag for a chosen message, even when allowed to manipulate half of the bits of the key k, query a single signature, and perform a polynomial number of verification queries, manipulating and changing half of the key bits they control. The advantage of an adversary A is the probability of forging an accepted tag in the attack game illustrated below.

MAC Attack Game

```
1: $k \leftarrow \texttt{MAC.KeyGen}(1^\lambda)$
2: $k_1 \| k_2 \leftarrow k$
3: $(msg^*, b, k_b, st) \leftarrow \mathcal{A}_1(1^\lambda)$    ▷ First execution of adversary. Its state can be stored in $st$.
4: if $b = 0$ then
5:     $k^* \leftarrow k_b \| k_2$
6: else
7:     $k^* \leftarrow k_1 \| k_b$
8: end if
9: $tag^* \leftarrow \texttt{MAC.Sign}(k^*, msg^*)$
10: $(msg', tag') \leftarrow \mathcal{A}_2^{\texttt{Verify}(\cdot,\cdot,\cdot)}(st, tag^*)$    ▷ Now adversary tries to produce a forgery.
11: return $\texttt{MAC.Verify}(k^*, msg', tag')$
```

Algorithm 9 Verification Query for MAC Attack Game

```
1: Verify($k_b, msg, tag$):
2: if $b = 0$ then
3:     $k' \leftarrow k_b \| k_2$
4: else
5:     $k' \leftarrow k_1 \| k_b$
6: end if
7: return $\texttt{MAC.Verify}(k', msg, tag)$
```

In the MAC attack game, the adversary sends a message in a query and chooses half the bits of the MAC key. It gets as response a computed tag for the queried message and with this information and the ability to verify MAC tags, it needs to produce a forgery. The adversary's advantage is the probability of successfully forging a valid tag.

2.4. Signature Schemes

A signature scheme consists of the following algorithms defined over sets (M, S), where M is the set of all possible messages, and S is the set of signatures:

- **Key Generation:** $SIG.KeyGen(1^\lambda) \rightarrow (pk, sk)$: A probabilistic algorithm that, given the security parameter 1^λ, generates a cryptographic key pair (pk, sk). Here, pk is the public key and sk the secret key;
- **Signing:** $SIG.Sign(sk, msg) \rightarrow sig$: A probabilistic algorithm that, given a secret key sk and a message $msg \in M$, produces a signature $sig \in S$;
- **Verifying Signature:** $SIG.Verify(pk, msg, sig) \rightarrow b$: A deterministic algorithm that, given a public key pk, a message $msg \in M$, and a signature $sig \in S$, returns a binary value b. If $b = 1$, the signature is valid (`accept`); otherwise, it is invalid (`reject`).

A tag generated by SIGN for a message is always accepted when verifying it with the same message and key.

The security of a signature scheme is based on the Unforgeability under the Chosen Message Attacks (UF-CMA) model. This model, formalized by Goldwasser *et al.*, the adversary needs to produce a forgery in the signature and is allowed to adaptively query for the signatures for any message. However, the forged signature produced by the adversary must be for a new message, not a message previously queried. The advantage of some adversary is the probability that it returns 1 in the attack game below:

Signature Attack Game

```
1: $(pk, sk) \leftarrow \texttt{SIG.KeyGen}(1^\lambda)$
2: $Q \leftarrow \emptyset$
3: $(msg^*, sig^*) \leftarrow \mathcal{A}^{SigningQuery(sk,\cdot)}(1^\lambda, pk)$
4: if $SIG.Verify(pk, msg^*, sig^*) = \texttt{accept}$ and $msg^* \notin Q$ then
5:     return 1
6: else
7:     return 0
8: end if
```

Algorithm 11 Signing Query for Sigature Attack Game

```
1: $\texttt{SigningQuery}(sk, msg)$:
2: $Q \leftarrow Q \cup \{msg\}$
3: return $\texttt{SIG.Sign}(sk, msg)$
```

A signature scheme is considered secure under the UF-CMA model if, for all adversaries $\mathcal{A}$, their advantage is always a negligible function of λ. This probability is denoted $Adv_{SIG}^{UF-CMA}(\mathcal{A})$.

2.4.1. ECDSA Signature Scheme

An example of a signature scheme is the Elliptic Curve Digital Signature Algorithm (ECDSA), defined over a group G derived from an elliptic curve. We treat G as a multiplicative group whose order is given by q and g is one of its generators. ECDSA signatures are defined over $\mathbb{Z}_q^2$ where q is the order of the elliptic curve and require a hash function $H : M \to \mathbb{Z}_q$ where M is the message space. The algorithms are:

Algorithm 12 ECDSA.Sign$(sk, msg) \to (r, s)$

```
1: $x \leftarrow sk$
2: repeat
3:     $k \leftarrow \mathbb{Z}_q$
4:     $y' \leftarrow g^k$
5:     Let $r$ be the $x$-coordinate of the point $y'$ on elliptic curve $G$ modulo $q$
6:     $s \leftarrow (H(msg) + rx)\, k^{-1} \pmod{q}$
7: until $r \neq 0$ and $s \neq 0$
8: return $(r, s)$
```

Algorithm 13 ECDSA.Verify$(pk, msg, (r, s)) \to b$

```
1: $y \leftarrow pk$
2: if $s = 0$ or $r = 0$ then
3:     return reject
4: end if
5: $a \leftarrow H(msg) \cdot s^{-1} \pmod q$
6: $b \leftarrow r \cdot s^{-1} \pmod q$
7: $y' \leftarrow g^a \cdot y^b \pmod q$
8: if $y'$ is the point at infinity in $G$ then
9:     return reject
10: end if
11: if the $x$-coordinate of $y'$ on $G$ modulo $q$ equals $r$ then
12:     return accept
13: else
14:     return reject
15: end if
```

2.5. Combiners and Hybrid Constructions

Key Encapsulation Mechanism (KEM) combiners offer a method for constructing a new KEM by integrating two or more existing KEMs. The principal security goal of KEM combiners is to ensure that the resulting combined KEM remains secure even if one of the original KEMs is compromised. This robustness is particularly valuable in developing hybrid schemes that combine classical and post-quantum algorithms, thereby enhancing resilience against future vulnerabilities.

Several approaches to KEM combiners have been explored in the literature. We mention two of them that meets the discussed security definition for KEMs:

- **XOR-then-MAC Combiner:** These operate by XORing the outputs of the combined KEMs to generate an initial shared secret. This secret is then authenticated using a Message Authentication Code (MAC) to protect against mix-and-match attacks, which could compromise the security of more naive combiners by allowing an adversary to mix parts of different ciphertexts to forge valid combinations.
- **Dual-PRF Combiner:** This combiner works running both KEMs to produce a pair of secrets (k_1, k_2) and a pair of ciphertexts (c_1, c_2). The pair of secrets are combined in a pseudo-random function to produce a key k that instantiates another pseudo-random function PRF. Then, we run $PRF(k, (c_1, c_2))$ to produce the final secret.

Both combiners are further analyzed and have their security proved in [11]. A comprehensive analysis of KEM combiners and their security properties is provided in the work of Giacon et al. [12], which formalizes the security definitions and constructions for these combiners. Their study highlights the importance of properly defining and constructing combiners to achieve desired security guarantees.

In our proposal, we specifically evaluate the performance and security of the XOR-then-MAC combiner.

2.6. The KEMTLS Protocol

KEMTLS is a proposed protocol designed to establish secure connections using post-quantum algorithms, with the potential to replace the current TLS 1.3 standard. At a high level, the protocol operates as follows [4]:

1. $C \to S$: "ClientHello"

   The Client sends a "ClientHello" message containing a newly generated ephemeral public key pk_e, created by running $KEM.KeyGen(1^\lambda)$.

2. $S \rightarrow C$: "ServerHello"

   The server responds with a "ServerHello" message that includes:

   - An encapsulated secret (c_e) generated by using the KEM's encapsulation function with the client's public key (pk_e);
   - A certificate chain that includes the server's public key (pk_S).

3. $C \rightarrow S$: "ClientKeyExchange"

   The client replies with a "ClientKeyExchange" message containing c_S, which encapsulates secret k_S. This encapsulation is computed using $KEM.Encaps(pk_S)$.

   The final shared secret is derived by combining k_e and k_S. With both known secrets, the client can send encrypted application data alongside the "ClientKeyExchange" message, even before the handshake is fully completed.
4. $S \rightarrow C$: "Finished"

   The server sends a "Finished" message to confirm the successful exchange of secrets. At this point, both parties have established the shared secret. Optionally, the server can send an encrypted application data response and the "Finished" message.

While the full handshake in KEMTLS requires two round trips, which may seem less efficient compared to the single round trip of TLS 1.3, this limitation can be effectively mitigated. The client has the option to send application data concurrently with the "ClientKeyExchange" message, even before the handshake is fully concluded, thereby reducing the overall latency.

Additionally, in scenarios where the client already possesses the server's public key, it can include the c_S encapsulation in the initial "ClientHello" message. This enables the server to respond with both the "ServerHello" and the "Finished" message within the second round trip, thereby achieving secret sharing in a single round trip.

The KEMTLS protocol also accommodates additional functionalities, such as client authentication. However, for clarity and focus, this paper addresses the fundamental protocol mechanics as described above. The principles introduced here can be readily extended to support these additional features as needed.

3. Our Hybrid KEMTLS Proposal

3.1. The Proposed Hybrid KEM Scheme

Combining public key KEMs (Key Encapsulation Mechanisms) with identity-based KEMs (IDKEMs) allows us to leverage the strengths of both cryptographic primitives. Our hybrid scheme integrates a PKI-based KEM with an identity-based ID-KEM using KEM combiners to create a robust and flexible hybrid KEM. The hybrid scheme consists of the following algorithms:

- $HKEM.MasterKeyGen(1^\lambda) \rightarrow (mpk, msk)$: This algorithm generates the master public key mpk and master secret key msk, identical to the $IDKEM.KeyGen$.
- $HKEM.KeyGen(1^\lambda) \rightarrow (pk, sk)$: Each user runs this algorithm locally to generate a public-private key pair (pk, sk), identical to $KEM.KeyGen$.
- $HKEM.Extract(msk, ID) \rightarrow sk_{ID}$: Given the master secret key msk and an identity ID, this algorithm generates a secret key sk_{ID} corresponding to the identity, identical to $IDKEM.Extract$.
- $HKEM.Encaps(mpk, pk, ID) \rightarrow (k, c)$: This probabilistic polynomial-time algorithm, given the master public key mpk, a public key pk, and an identity string ID, produces a secret k and a ciphertext c that encapsulates the secret securely.
- $HKEM.Decaps(sk_{ID}, sk, c) \rightarrow k$: This deterministic polynomial-time algorithm, given the secret key associated with identity sk_{ID}, another secret key sk, and a ciphertext c, recovers the secret k encapsulated in the ciphertext.

While the first three algorithms mirror those in the KEM and IDKEM schemes, we must define the encapsulation and decapsulation algorithms for hybrid KEM. Adopting the XOR-then-MAC construction from [12], we define them as follows:

Algorithm 14 $HKEM.Encaps(mpk, pk, ID)$

```
1: $(c_1, k_{1a} \| k_{1b}) \leftarrow \texttt{KEM.Encaps}(pk)$
2: $(c_2, k_{2a} \| k_{2b}) \leftarrow \texttt{IDKEM.Encaps}(mpk, ID)$
3: $k_{kem} \leftarrow k_{1a} \oplus k_{2a}$
4: $k_{mac} \leftarrow k_{1b} \| k_{2b}$
5: $c \leftarrow (c_1, c_2)$
6: $t \leftarrow \texttt{MAC}_{k_{mac}}(c)$
7: return $((c, t), k_{kem})$
```

Algorithm 15 $HKEM.Decaps(sk_{ID}, sk, (c, t))$

```
1: $(c_1, c_2) \leftarrow c$
2: $k_{1a} \| k_{1b} \leftarrow \texttt{KEM.Decaps}(sk, c_1)$
3: $k_{2a} \| k_{2b} \leftarrow \texttt{IDKEM.Decaps}(sk_{ID}, c_2)$
4: $k_{kem} \leftarrow k_{1a} \oplus k_{2a}$
5: $k_{mac} \leftarrow k_{1b} \| k_{2b}$
6: if $t \neq \texttt{MAC}_{k_{mac}}(c)$ then
7:     return $\perp$
8: else
9:     return $k_{kem}$
10: end if
```

If we use a post-quantum IBE-KEM like DLP-IBE [8], and combine it with a classical DHKEM, we can achieve a post-quantum construction without needing to modify existing keys and certificate chains, provided that the key in the leaf certificate is compatible with DHKEM (it must be an elliptic curve key, instead of an RSA key). We can then use KEMTLS to perform the handshake. Our construction based on HKEM will behave like a typical hybrid KEMTLS construction, with the following differences:

- We do not need to add a post-quantum key on the leaf certificate. In fact, we could keep using the same certificates we have nowadays, if they are compatible with DHKEM and if we assume that classical signatures are still secure. Therefore, compared to a typical hybrid KEMTLS, the data sent in our proposal during the "ServerHello" will be smaller, thanks to the smaller certificate chain. Based on the public key sizes for the finalists in the round 3 of NIST Post-Quantum standardization, this would represent an economy between 672 and 800 bytes (if using NIST security level 1) or between 930 and 1,134 bytes (if using NIST security level 3).
- In the "ClientKeyExchange" message, following the theoretical predictions from [13], we will need to send $N(2\lceil\log_2 q\rceil - l) + mlen + hlen$ bits as ciphertext produced by the post-quantum part of $HKEM$. In this case, N is the number of dimensions of the underlying lattice and must be at least 1024 to ensure 128 bits of security. The variable l is the compression parameter which increases negligibly the probability of decryption failure. It should be $l \leq \lfloor\log_2 q\rfloor - 3$ according with [8], where q is the modulus in the Ring-LWE setting. In the literature, suitable values for q have between 23 bits ([13]) and 28 bits ([14]). The variable $mlen$ is the size of the internal message encrypted by DLP-IBE and $hlen$ is the digest size for the hash used in the Fujisaki-Okamoto transform used to achieve IND-CCA security. Both of them would have 1280 (for 128 bits of security) or 1408 (for 192 bits of security).

  Therefore, the ciphertext in our proposal is expected to have between 3,488 bytes and 4,144 depending on the parameters suggested in the literature. By comparison, the ciphertext for NIST finalists have between 699 and 768 bytes (NIST security level 1), or between 930 bytes and 1088 bytes (NIST security level 3). This shows that unfortunately, using known realistic parameters, our proposal would produce a bigger ciphertext when compared to other KEM being standardized by NIST. The resulting increase in "ClientKeyExchange" message would be greater than the economy achieved in the "ServerHello" message.

- If the client do not know previously which PKG the server uses and do not know its master public key, then this information must be provided by the server. If needed, this information could be sent either inside the leaf certificate or, if we are not allowed to change existing certificate chains, the server could send it right after the certificate chain using an additional signed message. The size of the mpk is given by $N(\log_2 q)$ bits. Which mean that this requires between 2,944 and 3,584 additional bytes to be sent in the handshake, considering existing proposed values for N and q. If the mpk is not inside the leaf certificate, then it also requires an additional ECDSA signature to be verified using the key in the leaf certificate.

3.2. Security Modelling for the Hybrid Construction

Typically, combiners mix two constructions of the same cryptographic primitive to create a new construction of the same type. Our approach is different: we combine two related but distinct primitives to create a third, unique primitive that is neither a public-key KEM nor an identity-based KEM. Consequently, we need a new security definition for this hybrid primitive.

There are differences between the attack model between the classical security definition for PKI-based KEMs (Subsection 2.1) and identity-based KEMs (Subsection 2.2). Classical KEM security ensures that no adversary can break the security of a new key. In contrast, identity-based KEM security ensures that no adversary can break the security of any target chosen by them. The former is a single-user setting, while the latter is a multi-user setting. A strong security definition for public-key cryptography means that schemes secure in a single-user setting are also secure in the multi-user setting [15]. If the adversary wants information about different users, it can always simulate them producing their keys. However, this simulation cannot be achieved in identity-based encryption. Because of this, we need to allow the adversary to query oracles to better model its power in a real-world scenario. The same is true for our hybrid scheme; we should use identity-based KEM security as the basis for our hybrid construction.

There are notable differences between our hybrid model and the traditional identity-based KEM that must be incorporated into the security model. Each user in our model has an identity and a pair of local public and secret keys. Thus, when modelling the corresponding attack game, we must use a dictionary to store each pair of keys alongside their corresponding identities. Whenever an adversary queries information related to a specific identity, we must generate and store the associated keys in the dictionary, in case they do not exist.

Since public keys are not confidential, the adversary should be permitted to query the public key for any identity. Moreover, since an adversary in a real-world scenario could potentially corrupt or control other users, the model should allow queries to compromise selected identities. Following this query, the adversary obtains the local secret key sk and the identity secret key sk_{ID} associated with the identity.

This model also adapts decapsulation queries, enabling the adversary to specify the identity for which they seek decapsulation information.

The HKEM Attack Game together with Algorithms 17, 18 and 19 illustrates the complete definition of the attack game for our hybrid KEM.

HKEM Attack Game b (where b is either 0 or 1)

```
1: $Dict \leftarrow \texttt{EmptyDictionary}()$
2: $(msk, mpk) \leftarrow \texttt{HKEM.MasterKeyGen}(1^\lambda)$
3: $(ID^*, st) \leftarrow \mathcal{A}_1^{\texttt{Decaps}(msk,\cdot),\texttt{Compromise}(msk,\cdot),\texttt{PublicKey}(\cdot)}(mpk)$
4: $pk^* \leftarrow \texttt{PublicKey}(ID^*)$
5: $(c^*, k_0) \leftarrow \texttt{HKEM.Encaps}(mpk, pk^*, ID^*)$
6: $k_1 \leftarrow \mathcal{R}$                    ▷ Randomly chosen key from key space $K$
7: $b \leftarrow \mathcal{A}_2^{\texttt{Decaps}(msk,\cdot),\texttt{Compromise}(msk,\cdot),\texttt{PublicKey}(\cdot)}(st, c^*, k_b)$
8: return $b$
```

Algorithm 17 Decapsulation Query for HKEM Attack Game

```
1: Decaps($msk, ID, c$):
2: if $Dict[ID] \neq \emptyset$ then
3:     $(pk, sk) \leftarrow Dict[ID]$
4: else
5:     $(pk, sk) \leftarrow \texttt{HKEM.LocalKeyGen}(1^\lambda)$
6:     $Dict[ID] \leftarrow (pk, sk)$
7: end if
8: $sk_{ID} \leftarrow \texttt{HKEM.Extract}(msk, ID)$
9: return $\texttt{HKEM.Decaps}(sk_{ID}, sk, c)$
```

Algorithm 18 Compromise Query for HKEM Attack Game

```
1: Compromise($msk, ID$):
2: if $Dict[ID] \neq \emptyset$ then
3:     $(pk, sk) \leftarrow Dict[ID]$
4: else
5:     $(pk, sk) \leftarrow \texttt{HKEM.LocalKeyGen}(1^\lambda)$
6:     $Dict[ID] \leftarrow (pk, sk)$
7: end if
8: $sk_{ID} \leftarrow \texttt{HKEM.Extract}(msk, ID)$
9: return $(sk, sk_{ID})$
```

Algorithm 19 Public Key Query for HKEM Attack Game

```
1: PublicKey($msk, ID$):
2: if $Dict[ID] \neq \emptyset$ then
3:     $(pk, sk) \leftarrow Dict[ID]$
4: else
5:     $(pk, sk) \leftarrow \texttt{HKEM.LocalKeyGen}(1^\lambda)$
6:     $Dict[ID] \leftarrow (pk, sk)$
7: end if
8: return $pk$
```

The adversary's objective is to distinguish between both scenarios. Its advantage is $Adv^{IND-CCA}_{HKEM_{XtM}}(\mathcal{A}) = |\Pr[W_0] - \Pr[W_1]|$, where W_0 is the event in which it returns 1 in HKEM Attack Game 0, and W_1 is the event in which it returns 1 in HKEM Attack Game 1. The adversary cannot use ID^* as a Compromise Query or (ID^*, c^*) as a Decapsulation Query.

We can present and prove the following theorem with the security definition established, demonstrating that our hybrid construction utilizing the Xor-then-MAC combiner is secure.

Theorem 1. *Let $HKEM_{XtM}$ be a hybrid key encapsulation mechanism built using the Xor-then-MAC combiner with some message authentication code MAC, a public-key key encapsulation mechanism KEM and an identity-based key encapsulation mechanism IDKEM. Provided that for all adversaries $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, we have that $Adv_{KEM}^{IND-CCA}(\mathcal{A}_1)$, $Adv_{IDKEM}^{IND-CCA}(\mathcal{A}_2)$ and $Adv_{MAC}^{OT-sEUF}(\mathcal{A}_3)$ are all negligible values, then for all adversaries $\mathcal{A}_4$, $Adv_{HKEM_{XtM}}^{IND-CCA}(\mathcal{A}_4)$ is also a negligible value.*

The proof can be seen in Appendix A. We stress that our security proof follows the same structure and steps than the one from [12]. We just adapt it to our slightly different security definition.

3.3. The Full Proposal and General Security Discussion

By integrating our hybrid KEM into the KEMTLS protocol alongside another post-quantum KEM with ephemeral keys generated by the client, we can achieve a hybrid KEMTLS that does not require including a post-quantum key in the certificate. However, we still need to indicate which Private Key Generators (PKGs) the server supports, either extending the certificate or the protocol. The server may also provide a list of supported PKGs, allowing the client to select one it trusts when performing encapsulation with the hybrid KEM.

If we select a Diffie-Hellman KEM as the classical component of our hybrid construction, the public key in the certificate will be an elliptic curve point, similar to TLS 1.3. Consequently, in the case of a new certificate, the server can generate a Certificate Signing Request (CSR) using that elliptic curve point as the public key and submit it to any existing certificate authority.

CSRs require proof of knowledge by having applicants sign the message with the secret key corresponding to the public key being certified, typically using ECDSA. This ensures that the applicant possesses the corresponding secret key. However, this process poses a risk to our proposal. Unlike TLS 1.3, these keys are part of a KEM, not a signature scheme. Using the same keys for signing messages and later for secret exchange in a KEM may compromise the security of both the signature scheme and the KEM.

Although this operation cannot be executed using just any combination of signature scheme and KEM, we will demonstrate that employing the same pair of keys for an ECDSA signature and a DHKEM does not compromise the security of either cryptographic primitive. This analysis is challenging due to the lack of strong security guarantees for ECDSA signatures. Recent findings in [16] indicate that proving the security of ECDSA signatures without relying on idealized models like the generic group or random oracle models is not feasible. Nevertheless, literature presents various models for ECDSA security, which reduce the security of ECDSA to solving the discrete logarithm problem or similar problems.

We model the ECDSA signature as in [17], where only the line 5 of the signing algorithm, as described in Subsection 2.4.1, is replaced by an idealized function. Our analysis shows that the security proofs for ECDSA signatures and Diffie-Hellman KEM do not interfere with each other, and both remain valid even when these primitives share the same key.

We obtain this conclusion modelling the security with the help of two new SIG-KEM Attack Games: a Game 0 and Game 1. They combine the Signature Attack Game with the KEM Attack Games. Each one is modelled similarly as the two games that define IND-CCA security for the KEM, with the difference that now the adversary is able to query for signatures, assuming that signature and KEM share the same keys. In the end, the adversary outputs a bit b and a pair $(msg*, sig*)$. Let W_0 be the event where an adversary $\mathcal{A}$ outputs $b = 1$ in SIG-KEM Game 0 and W_1 the event where it outputs $b = 1$ in SIG-KEM Game 1. The advantage of a given adversary in the SIG-KEM game is either $|W_0 - W_1|$ or the probability of producing a valid forgery for a new message using the ECDSA signature in any of the two games, whichever value is greater.

SIG-KEM Attack Game 0

```
1: $(sk, pk) \leftarrow \texttt{KeyGen}(1^\lambda)$    ▷ Key generation shared between KEM and signature.
2: $Q \leftarrow \emptyset$    ▷ $Q$: set of messages sent to signing queries, initialized empty
3: $(c^*, k_0) \leftarrow \texttt{KEM.Encaps}(pk)$
4: $(b, msg*, sig*) \leftarrow \mathcal{A}^{\texttt{KEM.Decaps}(sk,\cdot),\texttt{SigningQuery}(sk,\cdot)}(pk, c^*, k_0)$
5: if $SIG.Verify(pk, msg^*, sig^*) = \texttt{accept}$ and $msg^* \notin Q$ then
6:     return $(b, 1)$
7: else
8:     return $(b, 0)$
9: end if
```

SIG-KEM Attack Game 1

```
1: $(sk, pk) \leftarrow \texttt{KeyGen}(1^\lambda)$
2: $Q \leftarrow \emptyset$
3: $(c^*, -) \leftarrow \texttt{KEM.Encaps}(pk)$
4: $k_1 \xleftarrow{\$} K$    ▷ Randomly chosen key from key space $K$
5: $(b, msg*, sig*) \leftarrow \mathcal{A}^{\texttt{KEM.Decaps}(sk,\cdot),\texttt{SigningQuery}(sk,\cdot)}(pk, c^*, k_1)$
6: if $SIG.Verify(pk, msg^*, sig^*) = \texttt{accept}$ and $msg^* \notin Q$ then
7:     return $(b, 1)$
8: else
9:     return $(b, 0)$
10: end if
```

Theorem 2. *Assuming that the computational Diffie-Hellman problem is hard, and that we can efficiently verify solutions for the problem, no adversary is able to have a non-negligible advantage in the SIG-KEM Attack Games when we instantiate the signature using ECDSA and the KEM using DHKEM.*

The proof can be seen in Appendix B. Notice that as our proposal involves a possible sharing of keys between these different primitives only in the classical part of the hybrid construction, the Diffie-Hellman assumption is acceptable, despite not being post-quantum.

Moreover, it is important to note that as shown in the proof, the information revealed by decapsulation queries does not interfere with the results or probabilities when computing responses for signing queries. Similarly, produced signatures do not interfere with decapsulation queries, thanks to using different random oracles for each query type. The upper bound for the probability of breaking the KEM or the ECDSA signatures when they share the same pair of keys is simply the sum of the probabilities of an adversary breaking the ECDSA and breaking the KEM.

Additionally, similar proofs can be derived using alternative models for ECDSA security. Some works, such as [18] and [19], prove the security of ECDSA signatures in the generic group model. Security proof for key sharing between ECDSA signatures and DHKEM could also be made in that model, but would require defining the gap Diffie-Hellman assumption in the generic model and extending the oracle to allow adversaries to query whether a triple of group elements is a Diffie-Hellman triple.

Another modeling of the ECDSA signature is presented in [20], where the security proof assumes that each message will not be signed twice (an assumption compatible with TLS). However, the security reduction is done to a weaker and non-standard assumption when compared with the computational Diffie-Hellman problem. Using that model, a similar security proof to the one in the appendix can be made, as that work simulates ECDSA signatures similarly. However, adapting that proof would require at the same time both the computational Diffie-Hellman assumption and the weaker intractable semi-logarithm assumption discussed in that article.

The security reduction from this last article applies not only to ECDSA and DSA signatures but also to Chinese SM2 [21] and Russian GOST [22] signatures, suggesting that our proposal could provide equivalent security for these signatures.

4. Experiments

To evaluate the performance and feasibility of our proposed hybrid KEMTLS, we modified the KEMTLS implementation from [23] to incorporate the hybrid construction detailed in Section 3. For the identity-based post-quantum KEM, we adopted the DLP-IBE scheme presented in [8], leveraging the implementation from [7]. The Circl library provided the DHKEM implementation for the classical KEM [24]. Additionally, we constructed a hybrid KEM combining Kyber [25] and DHKEM for ephemeral key encapsulation. The combiner logic and necessary KEMTLS adaptations were implemented in Go programming language [26].

Our proposal offers three distinct deployment scenarios based on the level of information exchange between the server and client.

- **KEMTLS-IBE-PDK:** In environments where the client possesses all necessary server keys, we employ KEMTLS with pre-distributed keys as described in [27]. This configuration enables a single-round handshake without requiring certificate chains or the IBE master public key (mpk);
- **KEMTLS-IBE:** Assuming the IBE mpk has been disseminated via out-of-band channels, we utilize the standard KEMTLS protocol;
- **KEMTLS-IBE-MPK:** For scenarios where the IBE mpk must be communicated during the handshake, we introduce an additional signed message following the certificate chain. The signing key is the P256 key embedded in the leaf certificate using ECDSA.

Our initial experiment aimed to assess the performance impact of transitioning from TLS 1.3 to our proposed solution without altering the underlying TLS infrastructure. To establish a baseline, we constructed a certificate chain comprising a self-signed RSA-4096 root certificate (using SHA-384), an intermediate RSA-2048 certificate (hashed with SHA-256), and a leaf certificate employing a 256-bit elliptic curve key (also hashed with SHA-256). This certificate chain configuration, representative of common practices on the internet, including Google's web pages, was used for our experiments.

To compare handshake sizes, we measured the total number of bytes exchanged between client and server, encompassing all messages. Table 1 presents the results. Note that the total handshake size slightly exceeds the sum of individual message sizes due to the inclusion of the final "Finished" message. The same certificate chain with size equal to 2050 bytes were used in all protocols, except in KEMTLS-IBE-PDK, as protocols with pre-distributed keys do not need certificates.

Table 1. Handshake message sizes (in bytes).

Protocol	ClientHello	ServerHello	ClientKeyExchange	Total
TLS 1.3	385	2,325	-	2,746
KEMTLS-IBE-PDK	5,291	974	-	6,301
KEMTLS-IBE	1,179	2,979	4,109	8,303
KEMTLS-IBE-MPK	1,179	6,639	4,109	11,963

To evaluate performance under simulated network conditions, we employed NetEM [28] on a local machine running Linux (Ubuntu 22.04.4) with an Intel i5 3.10GHz 8-core processor. We measured handshake time in milliseconds by timing the complete TLS handshake process. Network simulations imposed a 1000 Mbit/s bandwidth cap and varied packet delays between 0ms and 150ms. Table 2 summarizes the results, presenting mean and standard deviation values for handshake times across different protocols and latency settings. Additionally, the table includes measurements from a real-world network scenario for comparison.

Table 2. Handshake timings in miliseconds.

Delay	TLS 1.3		KEMTLS-IBE-PDK		KEMTLS-IBE				KEMTLS-IBE-MPK			
	Handshake Time		Handshake Time		Handshake Time		SendApp Time		Handshake Time		SendApp Time	
	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.
0	1.32	0.39	4.99	0.23	4.70	0.80	4.25	0.74	4.84	0.81	4.40	0.75
1	3.36	0.37	7.19	0.29	9.21	1.05	6.68	0.96	9.35	1.08	6.83	0.99
5	12.10	0.98	15.00	0.24	27.77	2.00	16.87	1.83	27.85	2.07	17.00	1.89
50	104.08	1.29	105.16	0.28	212.92	0.97	111.47	0.90	213.20	1.01	111.74	0.94
150	304.53	1.32	305.38	0.26	614.37	0.75	312.63	0.73	614.53	0.57	312.85	0.55
Real	97.74	0.84	99.23	0.70	197.28	1.69	100.49	1.30	197.51	1.66	100.70	1.05

Directly comparing handshake times between KEMTLS and TLS 1.3 is complex: KEMTLS has an additional round-trip, but both protocols can send application data directly after the first round-trip. To account for this, we use the concept of "Time-to-send-app-data," or just "SendApp Time,"introduced in [29], which measures the delay before the client can send application data. While both TLS 1.3 and KEMTLS-IBE-PDK exhibit similar timing for the full handshake and for the client sending application data, KEMTLS-IBE and KEMTLS-IBE-MPK demonstrate significantly shorter SendApp timings due to their ability to send application data before the end of the handshake process.

Figure 1 provides a visual representation of the handshake and SendApp time data using violin plots, allowing for a clearer understanding of the distribution of these values across different protocols and latency conditions. Each subplot represents the results for a specific network delay.

To conduct a more realistic evaluation, we deployed a client in London, UK, and a server in Dallas, USA, on virtual machines hosted by different providers. While the measured latency between these machines ranged from 40ms to 50ms, we acknowledge the dynamic nature of internet connections. We executed 1000 handshake iterations for each protocol, recording the full handshake time in milliseconds. The results of these real-world tests are incorporated into the final row of Table 2 and the last graph of Figure 1.

The data show that using our protocol, clients would need to wait between 2.6% and 7.1% more time before being able to send application data when connecting to a remote server with latencies compatible with a Internet connection ($\geq$ 50ms). The difference is bigger in low latencies, perhaps because the slower algorithms used by IBE have a bigger impact in such scenario. The data also show that at least in high bandwidth scenarios, sending or not an additional master public key (mpk) had little impact.

Next, to evaluate how our proposal compares with other hybrid constructions of KEMTLS, we prepared a second battery of tests that tried to replicate a scenario where all our current TLS infrastructure were already replaced to resist attacks made by quantum computers and the secret sharing and authentication in TLS now were always performed using KEMTLS protocol using hybrid schemes. We assumed that the certificate chains also would change to use hybrid algorithms for signatures and the leaf certificate now would optionally contain an IBE master public key (mpk) for KEMTLS-IBE-MPK.

To assess our proposal's performance relative to other hybrid KEMTLS constructions, we conducted a second set of experiments simulating a future TLS environment fully resistant to quantum attacks. In this scenario, TLS infrastructure is entirely replaced, with secret sharing and authentication exclusively handled by hybrid KEMTLS schemes. Certificate chains are modified to employ hybrid signature algorithms, and leaf certificates may optionally include an IBE master public key (mpk) for KEMTLS-IBE-MPK.

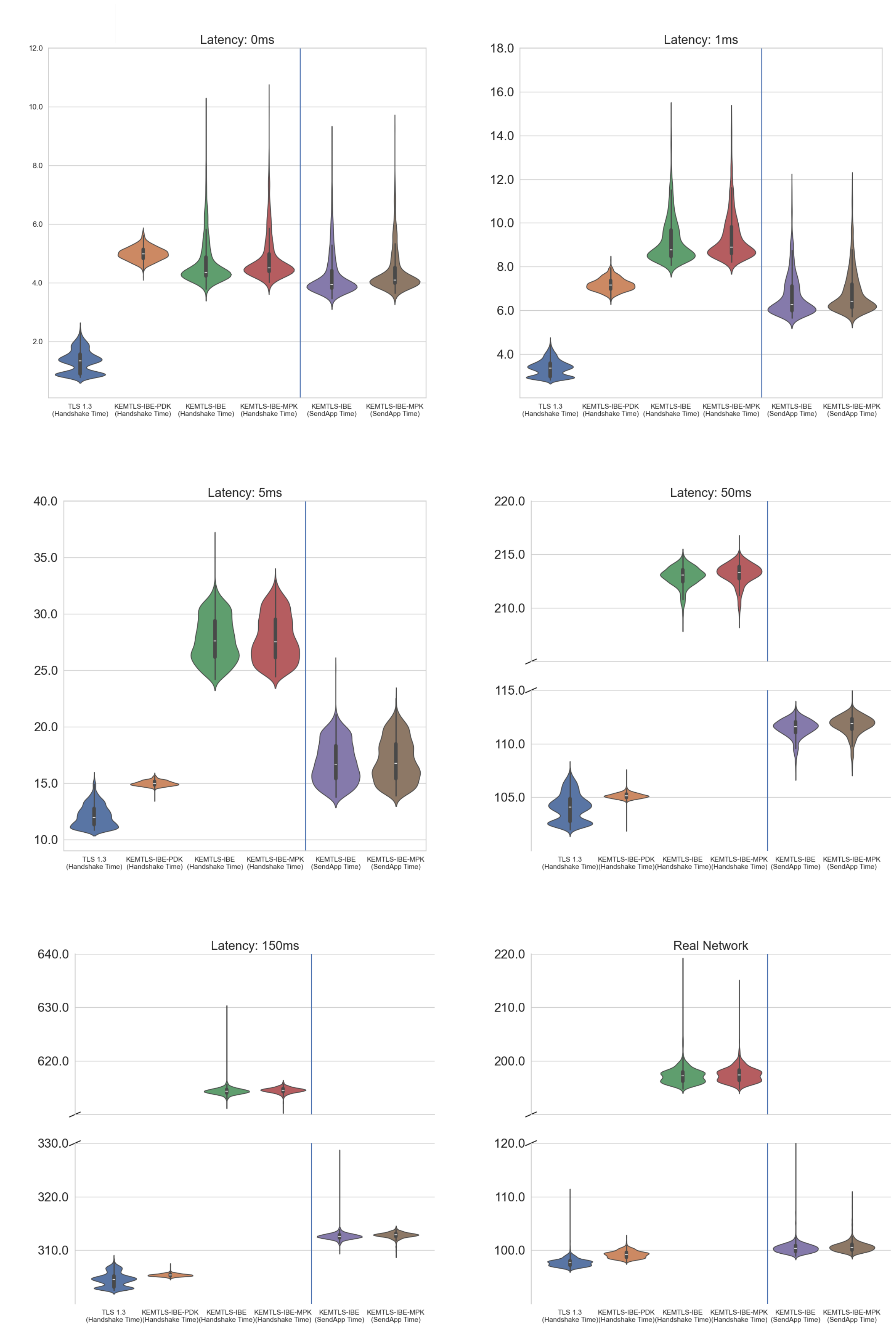

Figure 1. Violinplot containing a comparison of our proposal with TLS 1.3 under various circumstances.

To evaluate our proposal against other hybrid KEMTLS constructions, we selected several NIST Level 3 post-quantum KEMs for testing. This security level aligns with the DLP-IBE scheme's implementation constraints, which mandate a power-of-two value for the lattice dimension parameter N [8]. While a lattice dimension of 512 provides approximately 80-bit security (below NIST Level 1), increasing N to 1024 yields around 192 bits of security, suitable for NIST Level 3. Although some research, such as [13], explores modifications for 128-bit security, these changes minimally affect ciphertext size and have no impact on master public key length. Consequently, NIST Level 3 scenarios optimally leverage this IBE construction.

To comprehensively assess our proposal's performance within a fully post-quantum TLS environment, we constructed a second set of experiments. We combined various post-quantum KEMs with a classical DHKEM to create different hybrid KEMTLS configurations. To simulate a future post-quantum ecosystem, we incorporated post-quantum certificate chains employing a hybrid signature scheme combining ECDSA and the Dilithium-3 post-quantum signature [30]. Certificate chains were simplified to include only a root (omitted during transmission), an intermediate, and a leaf certificate. Table 3 presents the cumulative handshake message sizes for these configurations. This experimental setup aligns with the methodology outlined in [29] for comparative analysis.

Table 3. Handshake size (in bytes) comparing our Hybrid KEMTLS construction with traditional Hybrid KEMTLS protocols. All scenarios utilize a hybrid combination of DHKEM and a post-quantum KEM.

Hybrid Algorithms	ClientHello	ServerHello	ClientKeyExchange	Total
DHKEM + Saber	1,379	11,798	1,225	14,438
DHKEM + Kyber	1,571	11,994	1,225	14,826
DHKEM + NTRU HPS	1,317	11,581	1,067	14,001
DHKEM + NTRU-HRSS	1,525	11,996	1,239	14,832
KEMTLS-IBE	1,595	10,781	4,109	16,596
KEMTLS-IBE-MPK	1,595	14,384	4,109	20,124

Mirroring our previous evaluation, we conducted experiments simulating networks with varying latencies and packet loss on a local machine. Measurements comprised 1000 handshakes, with mean and standard deviation calculated for each scenario. Additionally, handshake timings were captured for a client and server geographically separated and connected over the internet.

Figure 2 presents violin plots comparing our proposal's handshake times with other hybrid KEMTLS algorithms under various latency conditions. Each subplot corresponds to a specific latency setting. Detailed data for all configurations can be found in Table 4.

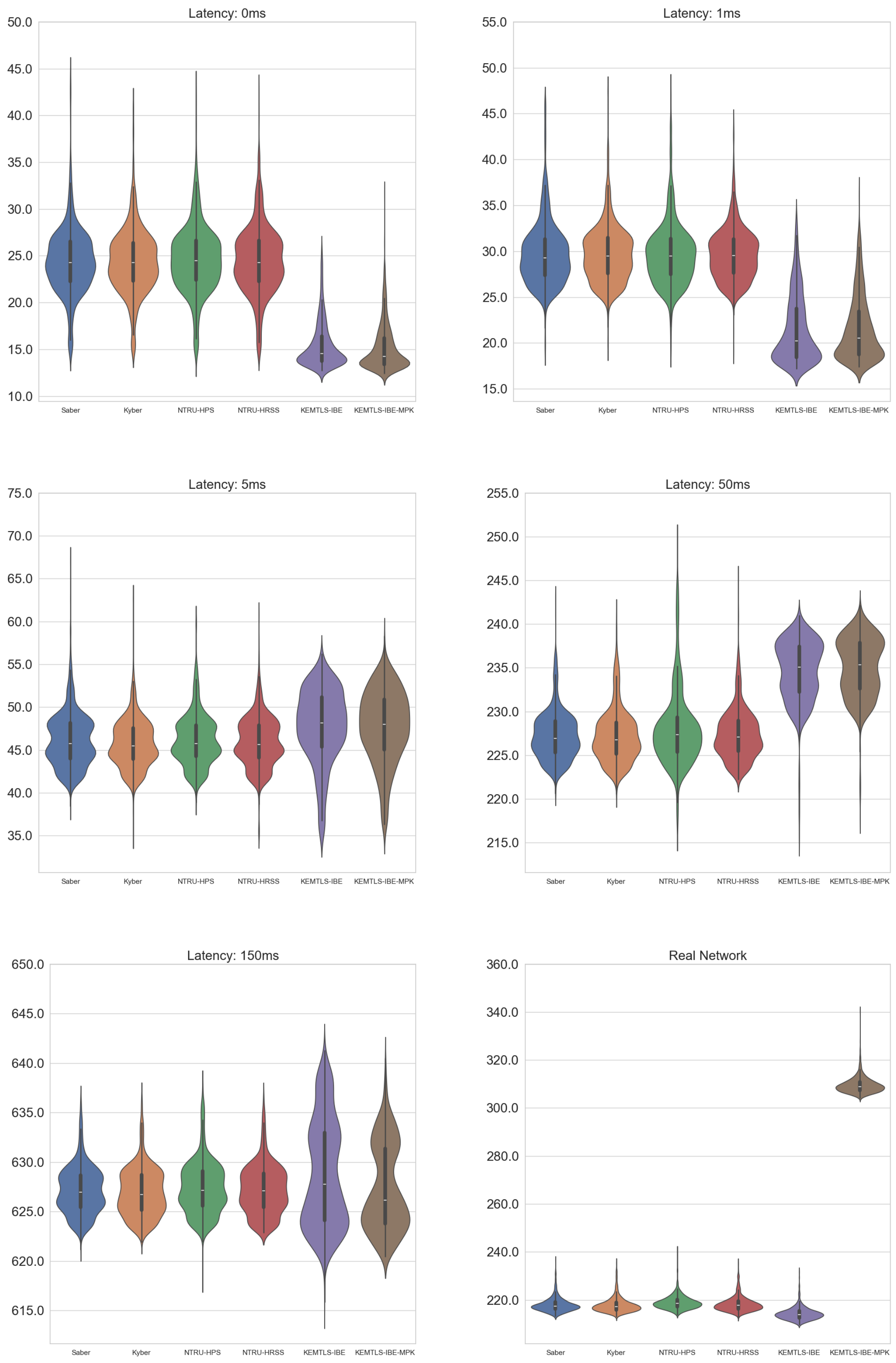

Figure 2. Comparison of our proposal with other hybrid algorithms in KEMTLS protocol.

Table 4. Handshake time (in milliseconds) for various hybrid KEMTLS configurations.

Latency	DHKEM+Saber		DHKEM+Kyber		DHKEM+NTRU-HPS		DHKEM+NTRU-HRSS		KEMTLS-IBE		KEMTLS-IBE-MPK	
	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.
0	24.43	3.60	24.39	3.47	24.60	3.65	24.55	3.65	15.47	2.48	15.18	2.45
1	29.76	3.47	29.68	3.06	29.73	3.47	29.68	2.81	21.46	3.68	21.56	3.50
5	46.17	3.14	45.72	2.82	46.14	2.96	45.94	2.89	47.70	4.38	47.67	4.25
50	227.24	2.76	227.24	3.06	228.06	4.39	227.39	2.88	234.61	3.53	235.06	3.39
150	627.14	2.38	626.96	2.43	627.41	2.65	627.22	2.48	628.82	5.32	627.38	4.34
Real	217.99	2.85	217.96	3.11	219.00	2.81	218.38	3.10	214.32	2.58	309.44	3.18

A surprising outcome was the lower handshake times observed for KEMTLS-IBE compared to other hybrid KEMTLS constructions, despite its larger message sizes. However, these performance differences were minimal, even smaller than variations observed in local, latency-free environments. This suggests that while network performance might be slightly degraded for KEMTLS-IBE, its computational efficiency compensates for this overhead, resulting in overall faster handshake times. Simulation data indicates that increased latency could potentially reverse this trend, favoring other hybrid algorithms.

Another interesting result is that despite not happening in simulation, when measured in a real network, KEMTLS-IBE-MPK performed much worse. We speculate that this happened because adding the mpk to the certificate crosses some threshold, creating a single message that is too big to be transferred with comparable speeds than the others.

We further evaluated the performance of KEMTLS-IBE-PDK, a variant utilizing pre-distributed keys, against other hybrid KEMTLS constructions with pre-distributed keys. Table 5 compares the handshake message sizes for these configurations while Table 6 and Figure 3 compare the timings.

Table 5. Handshake size (in bytes) comparing our Hybrid KEMTLS construction with traditional Hybrid KEMTLS protocols when using pre-distributed keys.

Hybrid Algorithms	ClientHello	ServerHello	Total
DHKEM + Saber (PDK)	2,601	1,331	3,963
DHKEM + Kyber (PDK)	2,793	1,326	4,185
DHKEM + NTRU HPS (PDK)	2,381	1,168	3,585
DHKEM + NTRU-HRSS (PDK)	2,797	1,376	4,209
KEMTLS-IBE-PDK	5,707	1,331	7,033

Table 6. Handshake time in milliseconds when the server keys were previously shared.

Latency	Saber (PDK)		Kyber (PDK)		NTRU-HPS (PDK)		NTRU-HRSS (PDK)		KEMTLS-IBE-PDK	
	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.	Mean	St.Dev.
0	22.55	3.18	22.13	2.91	22.38	3.08	22.60	3.20	15.26	2.47
1	25.26	3.13	25.67	3.14	25.48	3.09	25.72	3.25	22.66	4.88
5	34.82	2.92	35.19	2.87	35.20	2.95	35.44	2.94	40.26	4.66
50	128.64	4.03	128.55	4.21	128.87	4.25	129.37	4.23	127.19	4.79
150	326.82	2.63	326.40	2.26	326.95	2.35	327.06	2.56	323.71	1.79
Real	121.29	3.15	121.14	2.96	120.93	2.86	121.10	2.94	118.27	2.84

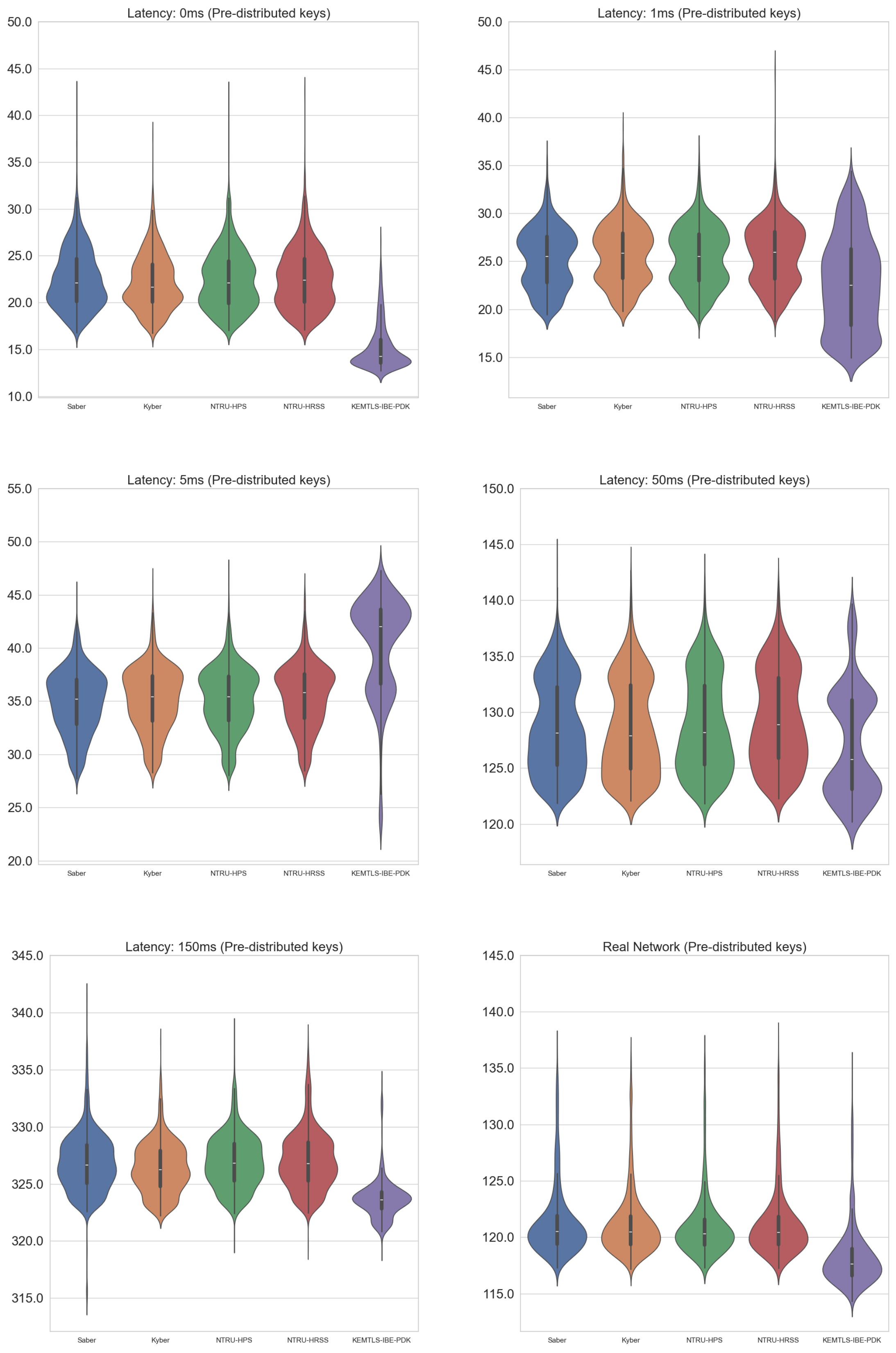

Figure 3. Comparison of our proposal with other hybrid algorithms in KEMTLS with pre-distributed keys.

Despite larger message sizes, our proposal demonstrated shorter handshake times in certain simulated and real-world network conditions. This unexpected outcome likely stems from the relatively minor impact of network latency compared to the computational efficiency of our implementation. In the pre-distributed key scenario, optimization differences between algorithms appear to play a more significant role in determining overall performance. To establish a more definitive comparison, further research is necessary to evaluate all algorithms under equivalent optimization levels.

5. Conclusion

This paper presents a hybrid Key Encapsulation Mechanism (KEM) for TLS 1.3, combining classical public key infrastructure (PKI) with identity-based encryption (IBE). Our approach is designed to smooth the transition to post-quantum cryptography by minimizing disruptions to existing TLS infrastructure and certificate management practices.

Summary of Contributions

- **Hybrid KEMTLS Protocol:** We introduce a novel hybrid KEMTLS protocol that seamlessly integrates traditional public-key cryptography and post-quantum identity-based encryption. This combined approach enhances security by providing resilience against both classical and quantum adversaries;
- **Practical Implementation:** Our implementation demonstrates the feasibility of achieving post-quantum security without requiring immediate modifications to existing TLS infrastructure. By leveraging identity-based encryption, we streamline key management and reduce the dependency on certificate authorities;
- **Rigorous Security Analysis:** We conducted a comprehensive security analysis, proving the resistance of our hybrid KEMTLS protocol to various attacks, including chosen-ciphertext attacks. The use of combiners ensures the overall security of the protocol even in the face of vulnerabilities in individual components;
- **Performance Evaluation:** Through extensive experimentation, we establish that our hybrid KEMTLS protocol delivers comparable performance to existing post-quantum solutions while minimizing disruptions to the current TLS ecosystem.

Implications for TLS Infrastructure

The integration of identity-based encryption into TLS 1.3 offers several practical advantages:

- **Minimal Infrastructure Impact:** Our hybrid approach requires minimal changes to the existing TLS infrastructure. By deriving keys from server identities, we eliminate the need for post-quantum key embedding in certificates, facilitating a smooth transition to post-quantum security;
- **Scalability and Compatibility:** The protocol seamlessly integrates with existing PKI systems, enabling organizations to adopt post-quantum security incrementally without a complete overhaul of their cryptographic infrastructure;
- **Enhanced Security Posture:** Combining classical and post-quantum cryptography fortifies the overall security posture. The protocol's resilience is ensured by its ability to withstand vulnerabilities in individual cryptographic components.

Future Work

Building on the foundations laid by this research, several directions for future work can be explored to further enhance and refine the hybrid KEMTLS protocol:

- **IBE Scheme Optimization:** Investigate techniques to enhance IBE scheme efficiency, focusing on reducing computational overhead and ciphertext sizes. Explore novel IBE constructions to improve overall protocol performance;

- **Integration of Emerging Post-Quantum Algorithms:** Expand the hybrid KEMTLS protocol to accommodate emerging post-quantum algorithms beyond IBE, such as lattice-based and code-based schemes, to create more versatile and resilient solutions;
- **Client Authentication:** Extend the protocol to support client authentication using identity-based signatures or other post-quantum signature mechanisms, thereby enabling comprehensive post-quantum security for both parties;
- **Scalability and Deployment Analysis:** Conduct in-depth studies to assess the protocol's performance and scalability in large-scale environments, including cloud and distributed systems. Identify potential bottlenecks and optimization opportunities;
- **Advanced Security Measures:** Explore the integration of multi-factor authentication and hardware-based security components to further strengthen the protocol's security posture;
- **Application Impact Assessment:** Evaluate the protocol's impact on existing TLS-dependent applications and services. Develop guidelines and best practices for smooth adoption and transition;
- **Interoperability with Classical TLS:** Design mechanisms to ensure seamless interoperability between the hybrid KEMTLS protocol and traditional TLS systems, facilitating gradual adoption and fallback options.

The proposed hybrid KEMTLS protocol represents a significant step forward in achieving post-quantum security within the TLS framework. By synergistically combining traditional PKI with identity-based encryption, we offer a practical and efficient solution for organizations to transition to a post-quantum future. This research not only addresses current security challenges but also lays a strong foundation for future advancements in secure communications.

Our research demonstrates that post-quantum cryptographic solutions can be seamlessly integrated into existing systems, facilitating widespread adoption. The proposed hybrid KEMTLS protocol, combining classical and post-quantum elements, offers a robust and scalable approach to securing digital communications in the face of evolving threats

Author Contributions: "Conceptualization, T.A. and R.C.; methodology, T.A. and R.C.; software, T.A.; validation, R.C.; formal analysis, T.A.; investigation, T.A.; resources, T.A. and R.C.; data curation, T.A.; writing—original draft preparation, T.A.; writing—review and editing, R.C.; visualization, T.A. and R.C.; supervision, R.C.; project administration, T.A. All authors have read and agreed to the published version of the manuscript.".

Funding: This study was financed in part by the Fundação de Amparo à Pesquisa e Inovação do Estado de Santa Catarina (Fapesc), granted by CP 48/2021.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data and source code for our experiments can be obtained in [31].

Acknowledgments: We would like to thank Alexandre Augusto Giron for suggestions about implementation. We also acknowledge the Technology Innovation Institute (TII) for its support.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A. Proof of Theorem 1

Proof: The proof has two steps: first we assume that our KEM is safer than our IDKEM and in the second step we assume the IDKEM is safer. From thee two steps we will conclude that in both scenarios, our hybrid KEM has a security similar to the strongest primitive. Meaning that even if one of them is broken, the other will keep the hybrid construction secure.

The proof in both steps proceeds using the standard hybrid argument, which is common in indistinguishable proofs: We show that an adversary cannot distinguish between two scenarios by building hybrid scenarios where we gradually transform from the first scenario to the second, and we show that this gradual change can be detected only if an adversary is able to break one of our assumptions.

Step 1: Assuming that KEM is Safer than IDKEM

Hybrid Game 2: This game is described below and using Algorithm 17. It lies between hybrid KEM Game 0 and hybrid KEM Game 1. In Game 2 we replace the real key produced by $KEM.Encaps$ by a random and uniform key, keeping the ciphertext c_1 intact. But we do not do the same for the key produced by $IDKEM.Encaps$. To produce consistent behavior, our decapsulation query now will produce the same random value every time it needs to decapsulate c_1.

Hybrid Attack Game 2 and 3

```
1: $ID^* \leftarrow \emptyset$
2: $Dict \leftarrow \texttt{EmptyDictionary}()$
3: $(msk, mpk) \leftarrow \texttt{HKEM.MasterKeyGen}(1^\lambda)$
4: $(ID^*, st) \leftarrow \mathcal{A}_1^{\texttt{Decaps}(msk,\cdot),\texttt{Compromise}(msk,\cdot),\texttt{PublicKey}(\cdot)}(mpk)$
5: $pk^* \leftarrow \texttt{PublicKey}(ID^*)$
6: $(c_1, k_{1a} \parallel k_{1b}) \leftarrow \texttt{KEM.Encaps}(pk^*)$
7: $(c_2, k_{2a} \parallel k_{2b}) \leftarrow \texttt{IDKEM.Encaps}(mpk, ID^*)$
8: $k_{1a} \parallel k_{1b} \xleftarrow{\$} K$                ▷ Randomly chosen key from key space $K$
9: $k_{kem} \leftarrow k_{1a} \oplus k_{2a}$
10: $k_{mac} \leftarrow k_{1b} \parallel k_{2b}$
11: $c \leftarrow (c_1, c_2)$
12: $t \leftarrow \texttt{MAC}_{k_{mac}}(c)$
13: $b \leftarrow \mathcal{A}_2^{\texttt{Decaps}(msk,\cdot),\texttt{Compromise}(msk,\cdot),\texttt{PublicKey}(\cdot)}(st, (c, t), k_{kem})$
14: return $b$
```

Algorithm 23 Decapsulation Query for Hybrid Attack Games 2 and 3

```
1: $\texttt{Decaps}(msk, ID, c)$:
2: if $Dict[ID] \neq \emptyset$ then
3:     $(pk, sk) \leftarrow Dict[ID]$
4: else
5:     $(pk, sk) \leftarrow \texttt{HKEM.LocalKeyGen}(1^\lambda)$
6:     $Dict[ID] \leftarrow (pk, sk)$
7: end if
8: $sk_{ID} \leftarrow \texttt{HKEM.Extract}(msk, ID)$
9: $(c'_1, c'_2) \leftarrow c$
10: $k'_{1a} \parallel k'_{1b} \leftarrow \texttt{KEM.Decaps}(sk, c'_1)$
11: if $c'_1 = c_1$ then                ▷ Only in Hybrid Game 2
12:     $k'_{1a} \parallel k'_{1b} \leftarrow k_{1a} \parallel k_{1b}$    ▷ Only in Hybrid Game 2
13: end if
14: $k_{2a} \parallel k_{2b} \leftarrow \texttt{IDKEM.Decaps}(sk_{ID}, c'_2)$
15: $k_{kem} \leftarrow k'_{1a} \oplus k_{2a}$
16: $k_{mac} \leftarrow k'_{1b} \parallel k_{2b}$
17: if $t \neq \texttt{MAC}_{k_{mac}}(c)$ then
18:     return $\perp$
19: else
20:     return $k_{kem}$
21: end if
```

Hybrid Game 2 and 3. The main difference between these Games and Hybrid KEM Game 0 for $HKEM_{XtM}$ is that in the challenge, we replaced the secret produced by KEM.Encaps with a value chosen randomly and uniformly. In Hybrid Game 2, we maintain consistency in the Decapsulation query by performing the tests in lines 12-13. In Hybrid Game 3, such consistency is not maintained.

Clearly, distinguishing between Hybrid KEM Game 2 and Hybrid KEM Game 0 is as hard as distinguishing between KEM Game 0 and KEM Game 1: if an adversary $\mathcal{B}$ could distinguish between both of them, it could be used to produce an adversary $\mathcal{A}_1$ that breaks the KEM security as described in Section 2.1. To show this, we build adversary $\mathcal{A}_1$ as follows:

- **Initialization:** Let p be the polynomial number of identities that we will interact with. We choose randomly one of them, initialize a counter and initialize $\mathcal{B}$:

```
1: procedure $\mathcal{A}_1(1^\lambda, pk, c*, k*)$
2:     $Dict \leftarrow EmptyDictionary()$
3:     $k \xleftarrow{\$} \{1, \ldots, p\}$
4:     $count \leftarrow 0$
5:     $(mpk, msk) \xleftarrow{\$} HKEM.MasterKeyGen(1^\lambda)$
6:     $\mathcal{B}(mpk)$
7: end procedure
```

- **Decapsulation Query:** Here we assume that $\mathcal{B}$ asks to decapsulate some value.

```
1: procedure $\mathcal{A}_1(msk, ID_i, c_i)$
2:     $(pk_i, sk_i) \leftarrow Dict.get(ID_i)$
3:     if no key was found in $Dict$ then
4:         $count \leftarrow count + 1$
5:         if $count = k$ then
6:             $Dict.store(ID_i, (pk, \emptyset))$
7:             $k_i \leftarrow Decaps(c_i[0])$                ▷ $\mathcal{A}_1$ runs its own decapsulation query
8:         else
9:             $(pk_i, sk_i) \xleftarrow{\$} HKEM.KeyGen(1^\lambda)$
10:            $Dict.store(ID_i, (pk_i, sk_i))$
11:        end if
12:    end if
13:    Now either we have the $KEM$ decapsulation $k_i$ for $c_i$ or we know the secret key $sk_i$ from
       which we can compute this decapsulation. Using this and $msk$ to compute the decapsulation
       for IDKEM, we can produce a valid response to $\mathcal{B}$.
14: end procedure
```

- **Compromise Query:** Here we assume that $\mathcal{B}$ asks to compromise some identity, revealing its secret keys.

```
1: procedure $\mathcal{A}_1(ID_i)$
2:     if no key was found in $Dict$ then
3:         $count \leftarrow count + 1$
4:         if $count = k$ then
5:             $\mathcal{A}_1$ cannot succeed and halts.
6:         else
7:             $(pk_i, sk_i) \xleftarrow{\$} HKEM.KeyGen(1^\lambda)$
8:             $Dict.store(ID_i, (pk_i, sk_i))$
9:         end if
10:    end if
11:    if $sk_i = \emptyset$ then
```

```
12:         $\mathcal{A}_1$ cannot succeed and halts.
13:     end if
14:     $sk_{ID_i} \xleftarrow{\$} HKEM.Extract(msk, ID_i)$
15:     return $(sk, sk_{ID_i})$
16: end procedure
```

- **Public Key Query:** If $\mathcal{B}$ queries for the public key for any identity, we get it from the dictionary or create it if it does not exist:

```
1: procedure $\mathcal{A}_1(ID_i)$
2:     $(pk_i, sk_i) \leftarrow Dict.get(ID_i)$
3:     if no key was found in $Dict$ then
4:         $count \leftarrow count + 1$
5:         if $count = k$ then
6:             $Dict.store(ID_i, (pk, \emptyset))$
7:             return $pk$
8:         else
9:             $(pk_i, sk_i) \xleftarrow{\$} HKEM.KeyGen(1^\lambda)$
10:            $Dict.store(ID_i, (pk_i, sk_i))$
11:        end if
12:    end if
13:    return $pk_i$
14: end procedure
```

- **Challenge:** When $\mathcal{B}$ chooses which identity ID_i will be the target for its attack. Without losing generality, we will assume that the identity already had its public key revealed by a Public Key Query. Otherwise, we could just simulate this query. This means that the identity has an existing public key in the dictionary:

```
1: procedure $\mathcal{A}_1(ID_i)$
2:     $(pk_i, sk_i) \leftarrow Dict.get(ID_i)$
3:     if $pk_i \neq pk$ then
4:         $\mathcal{A}_1$ cannot succeed and halts.
5:     end if
6:     $\mathcal{A}_1$ gets the challenge $(c*, k*)$ from its challenger
7:     return $(c*, k*)$
8: end procedure
```

- **Finalization:** When adversary $\mathcal{B}$ finalizes returning some bit b, adversary $\mathcal{A}_1$ also finalizes sending this same value b to its own challenger.

Notice that when adversary $\mathcal{A}_1$ is interacting with KEM Game 0, it produces responses for $\mathcal{B}$ compatible with Hybrid KEM Game 0. When $\mathcal{A}_1$ is interacting with KEM Game 1, it produces responses compatible with Hybrid KEM Game 2. Adversary $\mathcal{A}_1$ will produce a coherent output without halting only if it manages to guess the correct value for k during initialization, otherwise it will halt during the challenge or during some decapsulation query. Guessing correctly has a probability of $1/p$. Assuming that this is true, the probability of $\mathcal{A}_1$ succeeds are exactly the same that $\mathcal{B}$ succeeds. Therefore, if W_0 is the event where some adversary $\mathcal{B}$ outputs 1 in Hybrid KEM Game 0 and W_2 is the event where some adversary $\mathcal{B}$, we have for the corresponding adversary $\mathcal{A}_1$:

$$Adv_{KEM}^{IND-CCA}(\mathcal{A}_1) \geq \frac{1}{p}|Pr[W_2] - Pr[W_0]|$$

This result gives the negligible upper bound for distinguishing between those two games:

$$|Pr[W_2] - Pr[W_0]| \leq p \cdot Adv_{KEM}^{IND-CCA}(\mathcal{A}_1)$$

Hybrid Game 3: This hybrid Game is very similar to Hybrid Game 2. The main difference is that we no longer keep a consistent response in the Decapsulation Query when adversary tries to decapsulate the value c_1. Instead, if we are queried a value like this, we return the true decapsulation. In other words, we ommit the lines 10 and 11 described in the Decapsulation from Figure 8. Besides this change, the new Hybrid Game 3 is defined precisely as the Hybrid Game 2.

Notice that in Hybrid Game 3, the only way to distinguish the difference in the queries compared with Hybrid Game 2 is if the adversary can make a Decapsulation query with a pair where the second element is equal to the one present in the challenged ciphertext and the first element is different. It must also produce a valid tag that the MAC accepts; otherwise, in both Hybrid Games, the Decapsulation query would return $\perp$.

Therefore, if we had an adversary $\mathcal{B}$ that would be able to distinguish between Hybrid Game 3 and Hybrid Game 2, then we can also use it to create an adversary $\mathcal{A}_2$ that can win the MAC Attack Game (Subsection 2.3). We can build $\mathcal{A}_2$ as follows:

- **Initialization:** Adversary $\mathcal{A}_2$ will try to break the MAC security controlling the rightmost bits from the key, which are the bits produced by IDKEM (they cannot be considered independent random and uniform values in this Game, so it is important to control their bits in the MAC key). It follows the steps below:

  1. Use all the KEM, IDKEM and MAC algorithms to initialize adversary $\mathcal{B}$ with key mpk and to produce valid responses for all the queries performed by $\mathcal{B}$. It exchanges messages with $\mathcal{B}$ exactly like a valid challenger until $\mathcal{B}$ produces its challenge, sending ID as the target identity to attack. Let pk be the corresponding public key associated with such identity.
  2. Use the IDKEM and KEM algorithms to get an encapsulation $c*$ for such ID, getting also separately the secret k_2 produced by IDKEM and c_2, its correspondingly encapsulation.
  3. **return** $(ID, 1, k_2)$

- **Finalization:** $\mathcal{A}_1$ keeps interacting with $\mathcal{B}$ trying to produce a forgery. It does so performing the steps below:

  1. Sends $((c*, tag*), k*)$ as challenge to $\mathcal{B}$. Here, $c*$ is the encapsulation previously computed for ID, $tag*$ is the response got from the challenger and $k*$ is a random and uniform value.
  2. Keeps using all the KEM, IDKEM and MAC algorithms to produce valid and consistent responses for all the queries produced by $\mathcal{B}$, as long as it do not send any decapsulation query containing c_1.
  3. If we get a decapsulation query containing c_1 and a given tag, use the $Verification$ query to check the tag validity. If the tag is rejected, then return $\perp$ as response to the query. Otherwise, $\mathcal{A}_2$ halts and sends the tag as a successful forgery.

As discussed, the probability of $\mathcal{A}_2$ succeeding is the same as $\mathcal{B}$ getting a distinct behaviour between Hybrid Game 2 and Hybrid Game 3, which is the probability that it forges a tag for the MAC. Therefore, we get the following upper bound:

$$|Pr[W_3] - Pr[W_2]| \leq Adv_{MAC}^{OT-sEUF}(\mathcal{A}_3)$$

Finally, there is no difference between Hybrid Game 3 and KEM Game 1. In both cases, we return the true decapsulation for all queries, without trying to keep consistency with the encapsulation in the challenge. In Hybrid Game 3, the secret is computed with a XOR operation between a random and uniform value and some other value k_{2a} (as in Hybrid Attack Game 2 and 3, line 9). The result of such XOR is also a random and uniform value. Therefore, both games have no difference:

$$|Pr[W_1] - Pr[W_3]| = 0$$

Therefore, combining all these results, we have that for all adversaries $\mathcal{A}_4$, there are adversaries $\mathcal{A}_1$ and $\mathcal{A}_3$ such that:

$$Adv^{IND-CCA}_{HKEM_{XtM}}(\mathcal{A}_4) \leq p \cdot Adv^{IND-CCA}_{KEM}(\mathcal{A}_1) + Adv^{OT-sEUF}_{MAC}(\mathcal{A}_3)$$

Therefore, if KEM is safer than ID-KEM, we can use this upper bound, ignoring ID-KEM.

Step 2: Assuming that ID-KEM is safer than KEM

In the previous step, we shown that it is possible to ignore the security of ID-KEM and state the security of HKEM in terms only of the security of KEM and MAC. So, even if the ID-KEM is completely insecure, this do not necessarily compromise the security of HKEM.

In this step, we will show that it is also possible to state the security of HKEM using only the ID-KEM and MAC security, ignoring the KEM. Therefore, if the KEM is completely insecure, HKEM would still retain the security based on the other primitives.

The new Hybrid Games will be considered: Game 2′ and Game 3′. As they are not much different than Game 2 and Game 3, we describe them more briefly and show how we can use them to complete the proof.

Hybrid Game 2′: The main difference here is that instead of replacing the KEM secret (like in line 8, Hybrid Attack Game 2 and 3) by a random and uniform value, we do this with the ID-KEM secret (we do "$k_{2a}||k_{2b} \xleftarrow{\$} K$"). And then show that if some adversary could distinguish between Hybrid Game 2′ and Hybrid KEM Game 0, it could be used to build an adversary $\mathcal{A}_3$ that breaks the IND-CCA security for ID-KEM. The remaining of the proof is analogous, except that now adversary $\mathcal{A}_3$ do not need to guess which identity will be attacked. As both KEMs are in the multi-user setting, $\mathcal{A}_3$ can make its queries to reply for all queries presented by $\mathcal{B}$. Adversary $\mathcal{A}_3$ chooses which identity it will attack after the adversary $\mathcal{B}$ makes the same choice. Therefore, we get the following upper bound:

$$|Pr[W_2'] - Pr[W_0]| \leq Adv^{IND-CCA}_{IDKEM}(\mathcal{A}_3)$$

Hybrid Game 3′: As in Hybrid Game 3, here we update the Decapsulation query from Hybrid Game 2′ to always perform a proper decapsulation, not maintaining consistency with the random value that replaced the decapsulation of c_2 in Hybrid Game 2′. Again, it is possible to distinguish Game 2′ and Game 3′ only if an adversary can forge a tag in our MAC. The adversary $\mathcal{A}_3$, in this case, works as before, but it will choose to control the leftmost bits from the MAC key instead of the rightmost, as these values must be consistent with the secret k_1 produced by the KEM. This modification does not change the results; we have the same upper bound as before.

Combining both cases, we have that for all adversaries $\mathcal{A}_4$, there are adversaries $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$ such that:

$$Adv^{IND-CCA}_{HKEM_{XtM}}(\mathcal{A}_4) \leq MIN\left\{p \cdot Adv^{IND-CCA}_{KEM}(\mathcal{A}_1), Adv^{IND-CCA}_{IDKEM}(\mathcal{A}_2)\right\} + Adv^{OT-sEUF}_{MAC}(\mathcal{A}_3)$$

This upper bound concludes our security proof: it shows that given our assumptions, the upper bound for any attacker advantage in breaking our scheme is negligible. □

Appendix B. Proof of Theorem 2

Proof: As discussed in the text, we will use a modified KEM Attack Game where the adversary tries to distinguish between Game 0 and Game 1 or forge an ECDSA signature in either game. This augmented KEM attack game allows the adversary to query ECDSA signatures and assumes that the signature and the KEM (a DHKEM) share the same key.

Following the proof from [17], the operation on line 5 of the ECDSA signature algorithm (Subsection 2.4.1) is ideally modeled as a composition of functions $\psi \circ \Pi \circ \varphi$ that maps a group element to an integer modulo q. The function φ maps each elliptic curve point to a string encoding the x-coordinate of the point, Π is a bijective random oracle, and ψ performs a modulo q operation, where q is the elliptic curve group order.

First, we create a hybrid Attack Game 2 and show that if we had an adversary $\mathcal{B}$ able to forge signatures or distinguish between Attack Game 0 and Game 1, then we could solve the computational Diffie-Hellman problem using an adversary $\mathcal{A}$. Our adversary $\mathcal{A}$ will not act exactly like Game 0 or Game 1 from the KEM Attack Game. Instead, it will model a new Game 2, a hybrid between the two. Adversary $\mathcal{A}$ operates as follows:

- **Initialization:** Adversary $\mathcal{A}$ is initialized with the security parameter 1^λ and with a computational Diffie-Hellman instance $(G, g, q, y_a = g^a, y_b = g^b)$. Its objective is to find g^{ab}.

  Adversary $\mathcal{A}$ also initializes $\Pi^{\rightarrow}$ as an empty list that will store tuples (a, b) with $a \in A$ and $b \in B$ to simulate the bijective random oracle $\Pi : A \rightarrow B$. Another list, $\Pi^{\leftarrow}$, is also initialized as empty and will store values produced by backward queries to the bijective random oracle. It also initializes empty lists *DecapsList* and *OracleList*, which will help to produce consistent responses when simulating the random oracle for the KEM. A fifth list, *SignList*, is also initialized as empty and will store signatures produced during signing queries. A list of Q distinct values is also randomly sampled from set B: $\beta_1, \ldots, \beta_Q$, where Q is the expected number of queries to the bijective random oracle. Finally, a counter t is initialized to zero and will be incremented for each query to the bijective random oracle.

  Adversary $\mathcal{A}$ then initializes adversary $\mathcal{B}$, providing it with a randomness source ρ, sending 1^λ as the security parameter and $pk = y_a$ as the public key.
- **Bijective Random Oracle Query:** When $\mathcal{B}$ produces a query to check for $\Pi(a_i)$, adversary $\mathcal{A}$ checks if there is some $(a_i, b_i) \in \Pi^{\rightarrow}$ or $(a_i, b_i, \cdot) \in \Pi^{\leftarrow}$. If so, it sends b_i as a response. Otherwise, it increments the counter t, stores (a_i, β_t) in $\Pi^{\rightarrow}$, and sends β_t as a response.
- **Backward Bijective Random Oracle Query:** When $\mathcal{B}$ produces a query asking for $\Pi^{-1}(b_i)$, adversary $\mathcal{A}$ first checks if there is already some value $(a_i, b_i) \in \Pi^{\rightarrow}$ or some value $(a_i, b_i, \cdot) \in \Pi^{\leftarrow}$. If so, it sends a_i as a response. Otherwise, it chooses uniformly at random a new value $a_i \in A$ that is not present in $\Pi^{\leftarrow}$ or $\Pi^{\rightarrow}$. If a_i is not in the range of function φ, adversary $\mathcal{A}$ stores $(a_i, b_i, \perp)$ in $\Pi^{\leftarrow}$ and sends a_i as a response. Otherwise, if a_i is in the range of φ, it is discarded, and adversary $\mathcal{A}$ samples a random $\alpha \in \mathbb{Z}_q$ that is used to compute a new $a_i = \varphi(g^\alpha)$. The tuple (a_i, b_i, α) is stored in $\Pi^{\leftarrow}$ and a_i is sent as a response.
- **KEM Random Oracle Query:** When $\mathcal{B}$ sends a query asking for the KEM random oracle output for message m_i, we follow these steps:

  1. If $(m_i, k_i) \in OracleList$, then we send k_i as a response;
  2. We try to parse m_i as three points in the elliptic curve G. If this is not possible, we sample uniformly a random value k_i from the oracle range, store (m_i, k_i) in *OracleList*, and send k_i as a response;
  3. Let (y_1, y_2, y_3) be the three elliptic curve points parsed from m_i. If $y_3 \neq y_a$ or if (y_1, y_2, y_3) is not a Diffie-Hellman tuple, then we sample uniformly at random a value k_i from the oracle range, store (m_i, k_i) in *OracleList*, and send k_i as a response;
  4. If (y_1, y_2, y_3) is a Diffie-Hellman tuple and $y_3 = y_a$, we check if $y_2 = y_b$. If so, this means that $y_1 = y^{ab}$, and we have found a solution for the computational Diffie-Hellman problem. Otherwise, if $y_2 \neq y_b$, we check if there is a tuple (y_2, k_i) in *DecapsList*. If so, we send k_i as a response. Suppose we have not produced a response yet. In that case, we choose uniformly at random a value k_i from the oracle range, store (m_i, k_i) in *OracleList* and also store (y_2, k_i) in *DecapsList* to remember that k_i is the correct decapsulation for y_2 when using our public key value y_a. Then we send k_i as a response.

- **Signing Query:** If $\mathcal{B}$ sends a signing query with message msg_i, $\mathcal{A}$ can simulate a valid signature, even without knowing the secret key, as follows. First, it chooses uniformly at random some value b_i from the bijective random oracle range B. If $(\cdot, b_i) \in \Pi^O$, which happens with negligible probability, then $\mathcal{A}$ cannot produce a consistent signature and halts. Otherwise, it samples a random $s_i \leftarrow \mathbb{Z}_q$ and computes $y' \leftarrow g^{G(msg)s_i^{-1}} y_a^{\varphi(b_i)s_i^{-1}}$. Finally, it computes $a_i \leftarrow \phi(y')$ and checks if either a_i or b_i are already present in some tuple in Π^O. If so, $\mathcal{A}$ also fails to simulate a consistent bijective random oracle and halts. Otherwise, it stores (a_i, b_i) in $\Pi^{\leftarrow}$ and sends the signature $(\varphi(b_i), s_i)$ as the response to the query. If some invalid value is produced when creating the signature (for example, $r_i = \varphi(b_i) = 0$ or $s_i = 0$), adversary $\mathcal{A}$ can just try again, as done when creating a legitimate signature with the standard algorithm. All messages and signatures this query produces are stored in $SignList$.
- **Challenge:** To produce a challenge, adversary $\mathcal{A}$ generates the KEM challenge by choosing $y_b = g^b$ as the ciphertext c. Note that the correct secret should be the random oracle output for the input $g^{ab} \| g^b \| g^a$. Since it does not know the value of g^{ab}, instead, adversary $\mathcal{A}$ simply chooses uniformly at random some value k from the oracle range and sends (g^b, k) as the challenge. Notice that as the secret is chosen randomly and independently of the ciphertext, this behavior cannot be distinguished from Game 1, where the secret is also chosen randomly.
- **Decapsulation Query:** When $\mathcal{B}$ queries for the decapsulation of some $c_i \neq y_b$, adversary $\mathcal{A}$ checks if there is a tuple (c_i, k_i) in $DecapsList$. If so, it sends k_i as a response. Otherwise, $\mathcal{A}$ chooses uniformly at random some value k_i from the range of the random oracle, stores (c_i, k_i) in $DecapsList$, and sends k_i as a response.
- **Finalization:** After all queries, we assume that adversary $\mathcal{B}$ produces as output a bit trying to distinguish if it was in Game 0 or Game 1, and a possible forgery for the ECDSA signature.

As we mentioned, the Game that adversary $\mathcal{A}$ simulates is indistinguishable from Game 1, as the secret k sent during the challenge was chosen uniformly at random and is independent of the ciphertext $c = g^b$. However, given the nature of the random oracle, the only way to discover that the secret k sent during the challenge is random and not the decapsulation of g^b is by identifying that the oracle output for $g^{ab} \| g^b \| g^a$ (the real encapsulation of g^b) is different from k. The adversary $\mathcal{A}$ would discover this only if it sends an oracle query for $g^{ab} \| g^b \| g^a$. If it does so, it reveals to $\mathcal{B}$ the solution g^{ab} for the computational Diffie-Hellman problem. Therefore, even when using the keys to produce signatures, breaking the DH-KEM is still as hard as solving the computational Diffie-Hellman problem.

Let's assume we obtained a valid forgery $(msg^*, (r^*, s^*))$ for the ECDSA signature. From these values, adversary $\mathcal{A}$ can compute $y^* \leftarrow g^{G(msg^*)/s^*} y_a^{r^*/s^*}$. Therefore, let k be the discrete logarithm of y^*. We know from the exponents that $ks^* = G(msg^*) + ar^*$, even if we do not know the values of k and a. Now, it is possible to compute $a^* \leftarrow \varphi(y^*)$ and, without loss of generality, we can assume there is a pair in $\Pi^{\rightarrow}$ or in $\Pi^{\leftarrow}$ containing a^* as the first value. This could be because adversary $\mathcal{A}$ queried it from the bijective random oracle or because this value was produced during a signing query.

First, consider the case where $(a^*, b^*) \in \Pi^{\rightarrow}$ was produced during a signing query. This means that $\mathcal{A}$ also produced some other signature in the past in the form $(msg', (r', s'))$ such that $y' = g^{G(msg')/s'} y_a^{r'/s'}$ and such that $y' = g^{\pm k*}$. It can recover these values by searching in $SignList$. Therefore, it can obtain the equation $G(msg^*)/s^* + ar^*/s^* = \pm(G(msg')/s'^{-1} + ar'/s')$, from which $\mathcal{A}$ can also obtain a and then compute the solution $y_b^a = g^{ab}$ to the computational Diffie-Hellman problem.

In the second case, we have $(a^*, b^*, \alpha) \in \Pi^{\leftarrow}$. This means the tuple was produced by a backward bijective random oracle query. In this case, we know the value α such that $g^{\pm\alpha} = y^*$. From this, and the definition of y^*, $\mathcal{B}$ obtains the equation $\pm\alpha = G(msg^*)/s^* + ar^*/s^*$, which allows it to isolate the value of discrete logarithm a and compute the solution for the computational Diffie-Hellman problem.

Finally, in the third case, we have $(a^*, b^*) \in \Pi^{\rightarrow}$ produced by a bijective random oracle query. In this case, $b^* = \beta_j$ for some $0 < j \leq Q$.

Adversary $\mathcal{A}$ can then produce a new sequence of distinct values from B: $\beta'_j, \beta'_{j+1}, \ldots, \beta'_Q$ and execute adversary $\mathcal{B}$ again, using fresh new lists, giving the same source of randomness ρ and using the same input as before. The only difference is that when producing responses for the bijective random oracle if the counter t is incremented to some value $I \geq j$, we use the new value β'_i instead of the previous one. By the Forking Lemma, if $\mathcal{A}$ produces a forgery of this type with non-negligible probability, then with non-negligible probability, we get two forgeries from both executions such that the first produced a signature using β_j. The second uses β'_j (if this is not the case, $\mathcal{A}$ fails and aborts). Let $(msg^{**}, (r^{**}, s^{**}))$ be the second forgery. Combining the signatures from both executions, $\mathcal{A}$ can derive the equation $\pm(G(msg^*)/s^* + ar^*/s^*) = G(msg^{**})/s^{**} + ar^{**}/s^{**}$. The discrete logarithm a can be obtained from this, and the solution to the computational Diffie-Hellman problem can be computed.

The other two possibilities not mentioned above are that adversary $\mathcal{B}$ produces a forgery by finding a collision in the hash function G, or in the last equation, an adversary could produce values such that $G(msg^{**})/r^{**} = G(msg^*)/r^*$. Adversary $\mathcal{A}$ would fail to compute the discrete logarithm in these cases. Fortunately, these events have negligible probability if G is collision-resistant and has high division entropy resistance, which, as discussed in [17], is required to ensure the security of the ECDSA signature.

Therefore, except with negligible probability, $\mathcal{A}$ will not fail to simulate a bijective random oracle (we omit the calculus of such probabilities; they are the same found in ECDSA proof from [17]). If adversary $\mathcal{B}$ forges a signature, adversary $\mathcal{A}$ computes the solution for the computational Diffie-Hellman problem. Moreover, a solution is also computed with non-negligible probability if adversary $\mathcal{B}$ breaks the IND-CCA2 security of the DH-KEM with non-negligible probability. □

References

1. Rescorla, E. RFC 8446: The Transport Layer Security (TLS) protocol version 1.3, 2018.
2. Shor, P.W. Algorithms for quantum computation: discrete logarithms and factoring. Proceedings 35th annual symposium on foundations of computer science. Ieee, 1994, pp. 124–134.
3. of Standards, N.I.; (NIST), T. Post-Quantum Cryptography Project, 2024.
4. Schwabe, P.; Stebila, D.; Wiggers, T. Post-quantum TLS without handshake signatures. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 1461–1480.
5. Castryck, W.; Decru, T. An efficient key recovery attack on SIDH. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2023, pp. 423–447.
6. et al., D.J. SIKE - Supersingular Isogeny Key Encapsulation. https://sike.org/, 2021. Accessed: 19/06/2024.
7. Scott, M. On TLS for the Internet of Things, in a Post Quantum world. Cryptology ePrint Archive, Paper 2023/095, 2023. https://eprint.iacr.org/2023/095.
8. Ducas, L.; Lyubashevsky, V.; Prest, T. Efficient identity-based encryption over NTRU lattices. International Conference on the Theory and Application of Cryptology and Information Security. Springer, 2014, pp. 22–41.
9. Barnes, R.; Bhargavan, K.; Lipp, B.; Wood, C.A. Hybrid Public Key Encryption. RFC 9180, 2022. doi:10.17487/RFC9180.
10. Goldwasser, S.; Micali, S.; Rivest, R.L. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing* **1988**, *17*, 281–308.
11. Bindel, N.; Brendel, J.; Fischlin, M.; Goncalves, B.; Stebila, D. Hybrid key encapsulation mechanisms and authenticated key exchange. Post-Quantum Cryptography: 10th International Conference, PQCrypto 2019, Chongqing, China, May 8–10, 2019 Revised Selected Papers 10. Springer, 2019, pp. 206–226.

12. Giacon, F.; Heuer, F.; Poettering, B. KEM combiners. Public-Key Cryptography–PKC 2018: 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I 21. Springer, 2018, pp. 190–218.
13. Banerjee, U.; Chandrakasan, A.P. Efficient Post-Quantum TLS Handshakes using Identity-Based Key Exchange from Lattices. ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, 2020, pp. 1–6.
14. Güneysu, T.; Oder, T. Towards lightweight identity-based encryption for the post-quantum-secure Internet of Things. 2017 18th International Symposium on Quality Electronic Design (ISQED). IEEE, 2017, pp. 319–324.
15. Bellare, M.; Boldyreva, A.; Micali, S. Public-key encryption in a multi-user setting: Security proofs and improvements. Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19. Springer, 2000, pp. 259–274.
16. Hartmann, D.; Kiltz, E. Limits in the provable security of ECDSA signatures. Theory of Cryptography Conference. Springer, 2023, pp. 279–309.
17. Fersch, M.; Kiltz, E.; Poettering, B. On the provable security of (EC) DSA signatures. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1651–1662.
18. Brown, D.R. Generic groups, collision resistance, and ECDSA. *Designs, Codes and Cryptography* **2005**, *35*, 119–152.
19. Groth, J.; Shoup, V. On the security of ECDSA with additive key derivation and presignatures. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2022, pp. 365–396.
20. Fersch, M.; Kiltz, E.; Poettering, B. On the one-per-message unforgeability of (EC) DSA and its variants. Theory of Cryptography: 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II 15. Springer, 2017, pp. 519–534.
21. Chen, Z.; Liu, B.; Zheng, Y. The SM2 Cryptographic Algorithm. Technical Report GM/T 0003-2012, Chinese State Cryptography Administration, 2013.
22. on Technical Regulating, R.F.A.; Metrology. Information technology - Cryptographic data security - Signature and verification processes, 2012. GOST R 34.10-2012.
23. Celi, S.; Faz-Hernández, A.; Sullivan, N.; Tamvada, G.; Valenta, L.; Wiggers, T.; Westerbaan, B.; Wood, C.A. Implementing and measuring KEMTLS. Progress in Cryptology–LATINCRYPT 2021: 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6–8, 2021, Proceedings. Springer, 2021, pp. 88–107.
24. Cloudflare. CIRCL: Cloudflare Interoperable Reusable Cryptographic Library, 2024.
25. Avanzi, R.; Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber algorithm specifications and supporting documentation. *NIST PQC Round* **2019**, *2*, 1–43.
26. The Go Authors. The Go Programming Language. https://go.dev/, 2024.
27. Schwabe, P.; Stebila, D.; Wiggers, T. More efficient post-quantum KEMTLS with pre-distributed public keys. Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part I 26. Springer, 2021, pp. 3–22.
28. Hemminger, S.; others. Network emulation with NetEm. Linux conf au, 2005, Vol. 5, p. 8.
29. Giron, A.A.; do Nascimento, J.P.A.; Custódio, R.; Perin, L.P.; Mateu, V. Post-quantum hybrid KEMTLS performance in simulated and real network environments. International Conference on Cryptology and Information Security in Latin America. Springer, 2023, pp. 293–312.
30. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* **2018**, pp. 238–268.
31. Astrizi, T.L. Repository with the tests and source code. https://github.com/thiagoharry/go-kemtls, 2024.