# PQ-HPKE: Post-Quantum Hybrid Public Key Encryption

Mila Anastasova
*Computer Science and Engineering*
*Florida Atlantic University, USA*
manastasova2017@fau.edu

Panos Kampanakis
*Amazon Web Services*
USA
kpanos@amazon.com

Jake Massimo
*Amazon Web Services*
Seattle, USA
jakemas@amazon.com

***Abstract*—Public key cryptography is used to asymmetrically establish keys, authenticate or encrypt data between communicating parties at a relatively high performance cost. To reduce computational overhead, modern network protocols combine asymmetric primitives for key establishment and authentication with symmetric ones. Similarly, Hybrid Public Key Encryption, a relatively new scheme, uses public key cryptography for key derivation and symmetric key cryptography for data encryption. In this paper, we present the first quantum-resistant implementation of HPKE to address concerns that quantum computers bring to asymmetric algorithms. We propose PQ-only and PQ-hybrid HPKE variants and analyze their performance for two post-quantum key encapsulation mechanisms and various plaintext sizes. We compare these variants with RSA and classical HPKE and show that the additional post-quantum overhead is amortized over the plaintext size. Our PQ-hybrid variant with a lattice-based KEM shows an overhead of 52% for 1KB of encrypted data which is reduced to 17% for 1MB of plaintext. We report 1.83, 1.78, and 2.15 $\times 10^6$ clock cycles needed for encrypting 1MB of message based on classical, PQ-only, and PQ-hybrid HPKE respectively, where we note that the cost of introducing quantum-resistance to HPKE is relatively low.**

***Index Terms*—Post-Quantum, Hybrid Public Key Encryption, Post-Quantum Hybrid Public Key Encryption, Hybrid HPKE**

## I. Introduction

Public key cryptography is widely deployed in various use cases for key establishment, authentication and occasionally key encapsulation and data encryption. For example, (Elliptic Curve (EC)) Diffie-Hellman (DH) Key Exchange [21] is used to establish shared secrets among communication parties in protocols like TLS and SSH [45, 57, 37, 30]. RSA signatures [48] and (Elliptic Curve) Digital Signatures (DSA) [44, 36], on the other hand, are used for authentication.

However, there are shortcomings to public key algorithms, especially when used for data encryption. First, asymmetric cryptography is based on computationally expensive problems and introduces performance, power, and energy consumption challenges. Second, RSA, the most widely used asymmetric encryption option, introduces plaintext size limitations. It limits the data size it can encrypt to ranges of a few hundreds of kilobytes (KB) (e.g. 190B[1] when using RSAES-OAEP-SHA256 with RSA2048). It is for this reason that standalone asymmetric schemes are mostly used for encrypting small amounts of data such as symmetric encryption keys (key wrapping). Finally, an additional concern with asymmetric algorithms is quantum computers. Asymmetric algorithms rely on integer factorization (RSA) and (EC) Discrete Logarithm ((EC)DH, ECDSA) problems which could be broken by quantum computers in polynomial time using Shor's algorithm [52]. Thus, should a sufficiently powerful quantum computer become available, all public key cryptography used today could be broken, including key establishment and digital signatures. This brought the National Institute of Standards and Technology (NIST) to initiate the Post-Quantum (PQ) Cryptography Project [42] in order to standardize quantum-resistant key encapsulation mechanisms (KEMs) and signatures. Other standardization organizations have also been working on introducing post-quantum (PQ) algorithms to existing protocols and standards [56, 61] and focusing on post-quantum migration challenges and solutions [41].

Hybrid Public Key Encryption (HPKE) [10] is a recently ratified Internet Engineering Task Force (IETF) Informational RFC which leverages a KEM to establish a shared secret used to produce a symmetric key for symmetric encryption/authentication of the plaintext. Since HPKE provides a method of deriving symmetric keys using asymmetric key exchange mechanisms, data is encrypted using efficient algorithms such as AES-256-GCM which alleviates the computational complexity of asymmetric cryptographic primitives. Additionally, asymmetric encryption's previously limited plaintext size can be vastly increased. Therefore, similarly to other hybrid constructions, HPKE solves the plaintext size limitation and optimizes the computational cost of securing data with asymmetric algorithms.

To address the quantum computer risk, we introduce, to the best of our knowledge, the first implementation of a Post-Quantum Hybrid Public Key Encryption (PQ HPKE) scheme which uses quantum-resistant KEMs. Symmetric encryption is considered quantum-resistant, thus we leave HPKE's symmetric primitives intact. As PQ KEMs are relatively new and potentially not well-trusted yet, we offer PQ-only and PQ-hybrid HPKE ciphersuites. The latter combines a classical and a post-quantum KEM to generate the shared secret. If one of the KEMs is secure, the ciphertext is secure.

**Our Contributions:** Our contributions can be summarized as follows:

[1] $k-2hLen-16$, where $k$ is the RSA modulus and $hLen$ is the underlying hash function length[34].

1) We propose quantum-safe HPKE as a practical and secure quantum-resistant asymmetric encryption scheme.
2) We implement quantum-resistant HPKE by integrating two post-quantum KEMs in both PQ-only and PQ-hybrid modes.
3) We evaluate post-quantum HPKE's performance for various plaintext sizes and (a)symmetric encryption algorithms and compare it to classical HPKE and classical RSA encryption. We show that well-performing post-quantum KEMs are viable for use in HPKE.

**Use Cases:** HPKE is already used in some use cases to asymmetrically encrypt sensitive data. It is used in Message Layer Security (MLS) [8] to encrypt path secrets. MLS is a key establishment protocol developed in the IETF which enables group key establishment with forward secrecy and post-compromise security.

The ECH IETF draft [47] is also using HPKE to protect TLS client sensitive information like Subject Name Identifiers (SNI) which could reveal the destination the client is communicating with. ECH encrypts TLS *ClientHello*s with HPKE to a known TLS proxy's public key.

Other uses for HPKE include encrypting logging information [18, 17] and privacy preserving measurements of sensitive data [28]. It has also been proposed for Oblivious DNS over HTTPS (ODoH) [55]. Additionally, HPKE could replace current uses of RSA used for key wrapping or small plaintext encryption.

**Why HPKE**: We chose to focus on HPKE as it is the best asymmetric encryption option which could be integrated with PQ KEMs. While the NIST PQ Cryptography project is spearheading the standardization effort of quantum-resistant asymmetric algorithms, there is little discussion of how these algorithms will be incorporated into hybrid public key encryption schemes. Due to the nature of PQ KEMs, they could not operate asymmetric encryption like RSA or other classical algorithms, thus HPKE seems as the best candidate for asymmetric quantum-resistant encryption.

What's more, RSA has traditionally been used to asymmetrically encrypt small plaintexts. HPKE removes plaintext size limitations (dependent on modulus size), can provide authentication, and significantly improves its performance.

## II. Related Work

Symmetric and asymmetric algorithms have been combined in the literature in hybrid scheme variants [12, 33]. These usually combine symmetric primitives with an asymmetric algorithm as a symmetric key encapsulation mechanism. In [32], the authors study the performance of hybrid schemes.

Other, similar to HPKE, hybrid schemes have been proposed in the literature. [11] presents the Discrete Logarithm Augmented Encryption Scheme (DHAES) where two types of encryption based on RSA and Diffie-Hellman are defined. It is later renamed to Diffie-Hellman Augmented Encryption Scheme (DHAES) [1]. DHAES is based on a symmetric encryption algorithm, a message authentication code, and a cryptographic hash function. It was later referred to as Diffie-Hellman Integrated Encryption Scheme (DHIES) [2]. DHIES underwent several changes and was analyzed in the literature in [39, 26, 24, 25] where its variants received the common name ECIES [27]. ECIES was implemented in NaCL [60], Bouncy Castle [58] and Crypto++ [59]. It is standardized in non-interoperable standards [5, 35, 53], relies on deprecated or broken cryptographic primitives, and is not proven secure against chosen-ciphertext attacks (IND-CCA2) [9]; HPKE can offer authentication, uses modern primitives and benign malleability resistance instead.

What's more, the elliptic curve primitives used in HPKE have seen multiple platform-specific optimizations for AVX2 [29, 22, 16], ARM Neon [51] and ARM Cortex-M4 [23, 50, 49] which can speed up HPKE as well.

Additionally, [4, 38] offer an analysis of HPKE and discuss key compromise impersonation concerns in HPKE's authenticated mode.

The National Institute of Standards and Technology NIST [42] initiated a standardization process in 2016 to analyze, evaluate and optimize post-quantum algorithms. There are multiple research efforts optimizing the NIST PQ candidates summarized in [3, 40]. Upon standardization, post-quantum algorithms will be integrated into widely used cryptographic protocols such as TLS, SSH, and IKEv2. There are multiple studies evaluating quantum-safe algorithms in Internet protocols [14, 54, 43, 20].

The rest of this paper is organized as follows. Section III discusses HPKE and its underlying cryptographic primitives. Section IV presents how HPKE could be extended to become quantum-resistant and briefly discusses its security. Section V describes the setup and our experiments. In Section VI we analyze our results and quantitatively prove the post-quantum HPKE is viable, especially for well-performing KEMs. Section VII concludes this work and presents future topics of research.

## III. HPKE

### A. Cryptographic Primitives

**Key Encapsulation Mechanism (KEM)**: HPKE models key encapsulation as three steps:

- $\mathsf{KGen}() \rightarrow (pk, sk)$: A probabilistic key generation algorithm which generates a public key $pk$ and a secret key $sk$.
- $\mathsf{Enc}(pk) \rightarrow (ss, ct)$: A probabilistic encapsulation algorithm which takes as input a public key $pk$ and outputs a public encapsulation ciphertext $ct$ and shared secret $ss$.
- $\mathsf{Dec}(sk, ct) \rightarrow ss$: A deterministic decapsulation algorithm which takes a secret key $sk$ and the ciphertext $ct$ as input and returns a shared secret $ss$, or error.

HPKE instantiates the KEM as a ECDH KEM defined below:

- $\mathsf{KGen}() \rightarrow (pkE, skE)$: Generates an ephemeral public/private ECDH key pair.

- $\mathsf{Enc}(pkS) \rightarrow (ss, ct)$: Given input a static ECDH public key $pkS$, generates a ciphertext $ct$ and a shared secret $ss$:

$$ss \leftarrow ECDH(skE, pkS)$$
$$ct \leftarrow \text{SerializePublicKey}(pkE)$$

where *SerializePublicKey* is a KEM utility function that takes as input a public key $pk$ and produces a unique encoding. *DeserializePublicKey* reverses this process (i.e., *DeserializePublicKey(SerializePublicKey(pk)) = pk*).
- $\mathsf{Dec}(skS, ct) \rightarrow ss$: Given input a static ECDH private key $skS$ and *SerializePublicKey(pkE)*, deserializes $pkE$ and deterministically outputs a shared secret $ss$:

$$ss \leftarrow ECDH(skS, pkE)$$

In the rest of this work, we assume all transmissions of public keys include a *SerializePublicKey*led public key and we omit all *DeserializePublicKey* / *SerializePublicKey* operations.

**Key Schedule**: As it is also explained in [38, § 3.1], a key schedule is a tuple of deterministic algorithms (Hash, Extract, Expand) used for secret derivation and expansion. It is parameterized by a concrete hash function such as SHA-2, an input shared secret $i$, and an output length $N_h$.

- KDF($i$): A Key Derivation Function (KDF) computes the $N_h$-byte hash of $i$ by using the underlying hash algorithm.
- Extract($salt, label, IKM$): Generates a pseudorandom $N_h$-byte key $PRK$ by using input keying material $IKM$ with an optional string $salt$ and a label.
- Expand($PRK, label, info, L$): Generates $L$-byte pseudorandom string using the extracted key $PRK$ from the previous step, a label and optionally a string $info$.

**Authenticated Encryption with Associated Data (AEAD)**: As it is also explained in [38, § 3.1], an AEAD algorithm is a tuple of two steps ($Seal, Open$) defined over key, nonce, and message space $\mathcal{K}_{aead} = \{0,1\}^{8\times N_k}$, $\mathcal{N} = \{0,1\}^{8\times N_n}$, $\mathcal{M} = \{0,1\}^*$ respectively.

- $Seal(k, n, aad, m)$: Given key $k \in \mathcal{K}_{aead}$, nonce $n \in \mathcal{N}$, optional associated data $aad$, and plaintext message $m \in \mathcal{M}$, produces ciphertext (including an authentication tag) $c$. This is practically the encryption step.
- $Open(k, n, h, c)$: Given key $k \in \mathcal{K}_{aead}$, nonce $n \in \mathcal{N}$, optional associated data $aad$, and ciphertext $c$, produces the corresponding plaintext $m \in \mathcal{M}$, or error $\perp$ if decryption fails. This is practically the decryption step.

### B. HPKE Overview

HPKE is a relatively new IETF Informational RFC that replaced ECIES. In summary, it leverages (EC)DH to establish a shared key between two parties. The shared secret is used to derive a symmetric key using a *Key Schedule*. That symmetric key is then used to encrypt data with an efficient symmetric AEAD algorithm. HPKE specifies

- ECDH, X25519 and X448 as its KEMs
- HKDF-SHA256, HKDF-SHA384 and HKDF-SHA512 as its KDFs
- AES-GCM and ChaCha20/Poly1305 as its AEAD algorithms

The HPKE construction provides different operation modes for authenticating the sender of the data. In *Base mode* the sender is not authenticated. In *PSK mode* the sender is authenticated by a Pre-Shared Key (PSK). In *Auth mode* it is implicitly authenticated by its static private key used to establish the shared secret. In *AuthPSK mode* the sender is authenticated by a PSK and its static private key which are both used to establish the shared secret. In this work, we focus on *Base mode* as we are investigating post-quantum HPKE. *PSK mode* would not impact our results as it only adds a pre-shared key to the KDF. HPKE's two *Auth* modes cannot be implemented in PQ HPKE without introducing new messages from the receiver due to the nature of PQ KEMs being different than ECDH.

*Base mode* HPKE is shown in Figure 1. $S$ and $R$ mark the Sender and Recipient respectively. $E$ represents generated Ephemeral keys. We show an ECDH-based KEM. The recipient generates a static ECDH keypair ($skR, pkR$) where the public key is generated as the point multiplication of the secret key and a base point $B$ such that $pkR = skR \cdot B$. The public key $pkR$ is shared with the sender out-of-band. The sender generates an ephemeral ECDH keypair ($skE, pkE$) where $pkE = skE \cdot B$ for the same base point $B$. The sender then generates a shared secret $ss = skE \cdot pkR$. The ephemeral public key $pkE$ is then encoded and sent to the recipient. The recipient produces its view of the shared secret as $ss = skR \cdot pkE$. Both parties reach the same shared secret as $ss = skE \cdot pkR = skR \cdot pkE$. Following, a key derivation function KDF is applied to the shared secret. The output is a common key $ck$ which is input to the *Key Schedule Extract* and *Expand* steps to obtain the final symmetric key $k$ using a context variable.

The *Key Schedule* returns the symmetric key $k$, a nonce $n$, and an exporter secret $s_{exp}$. All these keys are only known to the sender who encapsulates the secret data and the recipient who decapsulates using the static private key. $k$ is used as the symmetric key. The nonce used in the *Seal* and *Open* operations is $n$ XORed with the current block counter. $s_{exp}$ can be used for exporting secrets of the desired length from the encryption context by using the corresponding KDF expand function, similar to the TLS 1.3 exporter interface [46].

The sender then proceeds to encrypt the plaintext to a ciphertext $c$ by using the *Seal* function with key $k$ and nonce from $n$ from its *Key Schedule*. The recipient decrypts the ciphertext $c$ by using *Open* with the key and nonce $n$ from its *Key Schedule*.

HPKE is formally analyzed in [38, 4] which show that in *Base mode* it is IND-CCA2 secure. In its authenticated modes, HPKE is Outsider-CCA and Insider-CCA secure.

## IV. Post-Quantum HPKE

After summarizing HPKE, we present its post-quantum version which uses quantum-safe KEMs to ensure that the ciphertext is protected against quantum computers. Note that HPKE is constructed with agility in mind. [10, § 9.1.3]

| **Alice (Sender)** | | **Bob (Receiver)** |
|---|---|---|
| Out of band knowledge of: | | |
| $pkR$ | | $(skR, pkR) \leftarrow \mathsf{KGen}()$ |
| | Encap | |
| $(skE, pkE) \leftarrow \mathsf{KGen}()$ | | |
| $ss \leftarrow ECDH(skE, pkR)$ | | |
| $ck \leftarrow KDF(ss, context)$ | $\xrightarrow{pkE}$ | |
| | Decap | |
| | | $ss \leftarrow ECDH(pkE, skR)$ |
| | | $ck \leftarrow KDF(ss, context)$ |
| | Key Schedule | |
| | $s \leftarrow \text{Extract}(ck, NULL)$ | |
| | $k \leftarrow \text{Expand}(s, \text{“hkpe key”}\|\|ctx, N_k)$ | |
| | $n \leftarrow \text{Expand}(s, \text{“hkpe nonce”}\|\|ctx, N_n)$ | |
| | $s_{exp} \leftarrow \text{Expand}(s, \text{“hkpe exp”}\|\|ctx, N_h)$ | |
| | Seal/Open | |
| $c \leftarrow Seal(k, n \oplus 0, aad, pt)$ | $\xrightarrow{c}$ | $pt \leftarrow Open(k, n \oplus 0, aad, c)$ |

Fig. 1: HPKE Overview

discusses the changes required to achieve post-quantum security. In this work, we propose and implement two versions of the HPKE construction. First we define a post-quantum-only HPKE where we replace the ECDH KEM with a PQ KEM. Second, we develop a PQ-hybrid HPKE which combines the classical ECDH KEM with a PQ KEM to generate the shared secret. The PQ-hybrid HPKE variant ensures that the ciphertext is secure in scenarios where the post-quantum algorithm is broken or a large-scale quantum computer has become available and threatens ECDH. Figure 2 shows PQ-only HPKE and PQ-hybrid HPKE in gray highlight.

### A. *PQ-only HPKE*

The PQ-only version of HPKE replaces the classical KEM with a PQ KEM. The rest of the scheme remains the same. Due to the nature of PQ KEMs, it does not involve an ephemeral key at the sender. The sender encapsulates a random PQ shared secret $ssS_{PQ}$ to the recipient's public key $pkR_{PQ}$ and sends the PQ ciphertext $ct_{PQ}$. The receiver decapsulates it using its private key $skR_{PQ}$ and produces $ssR_{PQ}$. At this point, both sides have the same shared secret $ss_{PQ} = ssS_{PQ} = ssR_{PQ}$ which is used to generate a common key $ck$ using a KDF as in classical HPKE. After the key is established, keys are derived from it by using the same *Key Schedule* as in classical HPKE (Figure 1). The plaintext is encrypted and decrypted using the derived key and nonce from the *Key Schedule*. Figure 2 shows the PQ-only HPKE variant in more detail excluding the gray highlighted text.

### B. *PQ-hybrid HPKE*

PQ-hybrid is a well-investigated concept that combines classical with PQ shared secrets to provide security against a quantum computer and a potentially broken PQ algorithm. [56] specifies PQ-hybrid key establishment for TLS 1.3. NIST, in [7], describes how a hybrid shared secret which is a concatenation of a classical shared secret generated by an approved method followed by an auxiliary shared secret are approved by NIST.

For our PQ-hybrid HPKE, we use the classical KEM along with a PQ KEM to generate two shared secrets. Figure 2 shows the PQ-hybrid HPKE variant in gray highlight. Similar to classical HPKE, the sender first generates an ephemeral keypair and a classical shared secret using the receiver's static public key $ssS_{ECDH} = skE_{ECDH} \cdot pkR_{ECDH}$. For the PQ part of the scheme, the sender encapsulates a random PQ shared secret $ssS_{PQ}$ to the recipient's public key $pkR_{PQ}$ and produces a ciphertext $ct_{PQ}$. It then sends its ephemeral ECDH public key and $ct_{PQ}$ to the receiver who produces the classical shared secret $ss_{ECDH} = ssR_{ECDH} = ssS_{ECDH} = skR_{ECDH} \cdot pkE_{ECDH}$ and decapsulates $ct_{PQ}$ by using its private key $skR_{PQ}$ to produce $ssR_{PQ}$. At this point, both sides have two shared secrets, $ss_{ECDH}$ and $ss_{PQ} = ssS_{PQ} = ssR_{PQ}$ which are concatenated and used as the shared secret. The resulting value is then fed to the KDF to obtain a common key $ck$. After the key is established, keys are derived from it by using the same *Key Schedule* as in classical HPKE (Figure 1). The plaintext is encrypted and decrypted using the derived key and

| **Alice (Sender)** | | **Bob (Receiver)** |
|---|---|---|
| Out of band knowledge of: | | $(skR_{PQ}, pkR_{PQ}) \leftarrow \mathsf{KGen}_{PQ}()$ |
| $\boxed{\text{pkR}}_{ECDH}, pkR_{PQ}$ | | $(\boxed{\text{skR}}_{ECDH}, \boxed{\text{pkR}}_{ECDH}) \leftarrow \boxed{\mathsf{KGen}}_{ECDH}()$ |
| | Encap | |
| $(\boxed{\text{skE}}_{ECDH}, \boxed{\text{pkE}}_{ECDH}) \leftarrow \boxed{\mathsf{KGen}}_{ECDH}()$ | | |
| $\boxed{\text{ss}}_{ECDH} \leftarrow ECDH(\boxed{\text{skE}}_{ECDH}, \boxed{\text{pkR}}_{ECDH})$ | | |
| $(ss_{PQ}, ct_{PQ}) \leftarrow \mathsf{Enc}_{PQ}(pkR_{PQ})$ | | |
| $ck \leftarrow KDF(\boxed{\text{ss}}_{ECDH} \| ss_{PQ}, context)$ | | |
| | $\xrightarrow{\boxed{\text{pkE}}_{ECDH},\, ct_{PQ}}$ | |
| | Decap | |
| | | $\boxed{\text{ss}}_{ECDH} \leftarrow \boxed{\text{ECDH}}(\boxed{\text{pkE}}_{ECDH}, \boxed{\text{skR}}_{ECDH})$ |
| | | $ss_{PQ} = \mathsf{Dec}_{PQ}(skR_{PQ}, ct_{PQ})$ |
| | | $ck \leftarrow KDF(\boxed{\text{ss}}_{ECDH} \| ss_{PQ}, context)$ |
| | Key Schedule | |
| | $s \leftarrow \text{Extract}(ck, NULL)$ | |
| | $k \leftarrow \text{Expand}(s, \text{"hkpe key"} \| ctx, N_k)$ | |
| | $n \leftarrow \text{Expand}(s, \text{"hkpe nonce"} \| ctx, N_n)$ | |
| | $s_{exp} \leftarrow \text{Expand}(s, \text{"hkpe exp"} \| ctx, N_h)$ | |
| | Seal/Open | |
| $c \leftarrow Seal(k, n \oplus 0, aad, pt)$ | $\xrightarrow{ct}$ | $pt \leftarrow Open(k, n \oplus 0, aad, c)$ |

Fig. 2: PQ-only and PQ-hybrid HPKE Overview

nonce from the *Key Schedule*.

### C. PQ HPKE Security

A hybrid scheme (excluding authentication modes) can be defined as a KEM-DEM construction, where the Key Encapsulation Mechanism encapsulates the symmetric key and the Data Encryption Mechanism (DEM) encrypts the plaintext. As it is also explained in [10], Cramer and Shoup [19] and Herranz et al. [31] proved that the KEM-DEM hybrid schemes are IND-CCA2 secure if and only if the underlying KEM and DEM schemes are IND-CCA2 secure.

As discussed in [10, § 9.1.2], the difference for HPKE is that its *Base mode* introduces additional KDF invocations which would have to be proven to be IND-CCA2 secure. That is done in [38, 4] which formally analyze classical HPKE. They show that in *Base mode* HPKE is IND-CCA2 secure. In its authenticated modes HPKE is Outsider-CCA and Insider-CCA secure by assuming Gap Diffie-Hellman property for the ECDH KEM and modeling the KDF Extract function as a random oracle and the Expand function as a PRF. Practically, the exported keys and the encrypted data are protected only if the receiver static KEM public key and the optional pre-shared key are safe. In other words, HPKE's authenticated modes are vulnerable to key-compromise impersonation if the optional pre-shared key and the recipient's KEM public key are revealed. Protections against such attacks are discussed in [10, § 9.1.1].

PQ-only and PQ-hybrid HPKE would require more analysis to be proven IND-CCA2 secure. Based on [19], *Base mode* PQ-only HPKE will still be IND-CCA2 secure as long as the PQ KEM is IND-CCA2 secure. PQ-only HPKE would still suffer from key impersonation attacks as the classical HPKE.

PQ-hybrid HPKE's security proof, on the other hand, would require more work. If we focus on *Base mode* HPKE, based on [19], we could prove it is IND-CCA2 secure if we proved that combining classical and PQ secrets is IND-CCA2 secure given that the AEAD and KDF properties in classical HPKE hold. [15] discusses the practical and theoretical security of combining classical and PQ secrets. It showed that the basic concatenation combiner (also used in our PQ-hybrid HPKE implementation) is secure in the random oracle model when the KDF is modeled as a random oracle and at least one of the KEMs is IND-CPA secure. The concatenation combiner could be proven to be IND-CCA2 secure if the ciphertext is given to the KDF as `context`. We did not do that into our implementation, but it would be trivial to add. More work would be required to prove the shared secret combiner used in HPKE is IND-CCA2 secure. Other combiners than simple concatenation could also be considered [13].

For its authenticated modes (*PSK*, *Auth*, *AuthPSK*), PQ-only and PQ-hybrid would require more work to prove that they are Outsider-CCA and Insider-CCA-secure like classical HPKE, That could include developing a new post-quantum hybrid authenticated KEM to use in the existing proof [4] or combining PQ KEM and signatures.

## V. Experiments

In this work, we evaluate PQ HPKE's performance and compare it with its classical counterparts. We ran all our experiments on an AWS EC2 instance based on Intel(R) Core(TM) i7-10610U CPU with 8GB of RAM running @1.80GHz with maximum turbo frequency of up to 4.90 GHz. The target processor featured 4 cores and 8MB of cache memory. For our implementations we used the AWS-LC cryptographic library [6].

Due to its wide adoption for encrypting small plaintexts or as a key wrapping mechanism we first evaluated RSA. The RSA instantiations we measured are PKCS#1.5, RSAES-OAEP and plain RSA without padding for 2048, 3072 and 4096-bit modulus. The RSA plaintexts were limited by the RSA modulus size. We then evaluated classical and PQ HPKE. The symmetric key algorithms we used are AES-GCM and ChaCha20/Poly1305. The plaintext sizes we considered were from 1KB to 1MB.

For PQ HPKE, we tested Kyber, one of NIST's Round 3 finalist KEMs which offers high performance, and SIKE, one of NIST's Round 3 alternate candidate KEMs which offers small keys and ciphertexts. We used the lowest security level parameter for each scheme, specifically, Kyber-512 and SIKEp434. We chose Kyber because, as a lattice scheme, it offers a balance between performance and public key and ciphertext sizes. Any of the other NIST Round 3 PQ KEM finalists would perform similarly to Kyber. For SIKE, its public key and ciphertext sizes are the smallest of all PQ KEM candidates. Thus, it could be used in use cases where memory or bandwidth resources are scarce. However, SIKE's elliptic curve isogeny maps result in slow performance. Our implementations included AVX2 assembly optimizations for Kyber and x64 optimizations for SIKE.

We measured *Key Generation* (KG), *Encryption* (E) / *Seal* (s) and *Decryption* (D) / *Open* (o) and HPKE *Setup Sender* (S) and *Setup Recipient* (R). *Setup Sender* and *Setup Recipient* include KEM encapsulation and decapsulation respectively. *Seal* and *Open* are used for symmetric encryption and decryption whereas *Encryption* and *Decryption* are used for RSA only. Our measurements did not include the HPKE receiver static ECDH and PQ key generation as that takes place offline.

We ran our experiments for each algorithm 1,000 times. To increase our accuracy, we eliminated the first and fourth quartile of our measurements. Additionally, all our results include the mean of the measured algorithm in CPU clock cycles.

We note that in PQ-hybrid HPKE, the physical location of the classical and PQ secrets in memory is important since it may impact the cache miss rate and the scheme latency on low-end target platforms which feature extra clock cycles for memory accessing instructions. For this reason, in our implementation, we stored both parameters ($ss_{ECDH}$ and $ss_{PQ}$) as a concatenation in a single variable and accessed them by changing the memory address offset. That reduces the cost of memory access because the secrets are in consecutive memory addresses.

Our results are presented in section VI.

## VI. Performance Evaluation

Below we present and analyze the experimental results of our performance measurements of RSA, classical and PQ HPKE as described in section V. They show that PQ HPKE performs satisfactorily especially for well performing PQ KEMs.

Table I shows our RSA performance results. We observe that key generation is RSA's costliest operation. Of course, as RSA keys are static, they can be considered generated offline. In terms of encryption and decryption of small plaintext and ciphertext sizes, encryption is very efficient whereas decryption requires significantly more cycles. As we see below, RSA decryption is much slower than PQ HPKE when Kyber is the PQ KEM.

| **Key Size [b]** | **Padding Mode** | **Size [B]** | **Cycles** ($\times 10^3$) **KG** | **E** | **D** |
|---|---|---|---|---|---|
| 2048 | PKCS #1.5 | 245 | 107,521 | 48 | 1,711 |
| | RSAES-OAEP | 214 | 88,963 | 51 | 1,715 |
| | NO PAD | 256 | 255,425 | 47 | 1,711 |
| 3072 | PKCS #1.5 | 373 | 1,105,263 | 101 | 5,226 |
| | RSAES-OAEP | 342 | 412,527 | 108 | 5,232 |
| | NO PAD | 384 | 737,968 | 101 | 5,223 |
| 4096 | PKCS #1.5 | 501 | 3,688,616 | 171 | 11,753 |
| | RSAES-OAEP | 470 | 2,433,587 | 180 | 11,763 |
| | NO PAD | 512 | 1,195,902 | 170 | 11,749 |

TABLE I: RSA performance for different plaintext sizes.

Figure 3 compares classical HPKE to PQ HPKE performance. More specifically, Figure 3a shows how the total performance (encapsulation, decapsulation, encryption, decryption) of classical HPKE with X25519 as the KEM compared to PQ-hybrid with X25519 and Kyber or X25519 and SIKE as the KEMs. We see that `X25519_SIKE` is ~49 times more costly than classical HPKE and around 32× slower than `X25519_Kyber` for 1KB plaintexts. SIKE's performance is significantly worse than X25519 and Kyber, so we expect its impact to be higher on small plaintexts. Even for bigger 1MB plantexts we see that SIKE's slow performance is not drastically amortized over the size of the plaintext. Specifically, it remains 11 and 9 times heavier than classical and PQ-hybrid `X25519_Kyber` respectively. The reason is that SIKE's performance is much slower than all other KEMs and symmetric operations of the scheme.

To study closer the X25519 and Kyber variants which perform better, Figure 3b compares classical X25519 HPKE with PQ-only HPKE with Kyber and PQ-hybrid with X25519 and Kyber KEMs. We see that `X25519_Kyber` HPKE is only $\sim$1.5 times heavier than classical X25519 HPKE for small 1KB plaintexts. For 1MB plaintexts it is only $\sim$1.2 times heavier as the PQ performance overhead is amortized over the plaintext size. Kyber-based PQ-only HPKE, on the other hand, is $\sim$1.9 times faster for 1KB plaintexts than classical X25519 HPKE mainly because optimized Kyber performs much faster than X25519 in software. That drops to $\sim$1.03 times as the symmetric operations cost increases for 1MB plaintexts.

We then focused on the breakdown of the operations for the best performing variants. Figure 4 shows the asymmetric and symmetric primitives cost broken down for classical HPKE with X25519 KEM and PQ-hybrid HPKE with `X25519_Kyber` for various plaintext sizes. We can see that the KEM cost is constant which is expected since encapsulation and decapsulation are used only once to establish the shared secret. Onward we see that as the size of the plaintext increases the symmetric encryption and decryption cost increases because the more data we encrypt, the more symmetric encryption takes. That also explains how the $\sim$1.5 times more expensive PQ-hybrid option for 1K plaintexts drops to almost the same cost when the plaintext increases to 1MB. Considering the magnitude of the KEM cost overall, we note that Kyber's performance still keeps the KEM to a satisfactory range. SIKE on the other hand would increase the KEM cost much more significantly. For comparison, Figure 4 includes a red line which depicts the encryption cost of RSAES-OAEP (2048-bits) for small 214B plaintexts which shows how much more expensive quantum-vulnerable RSA is compared to classical and PQ-hybrid HPKE with Kyber.

Now we compare RSA with classical and PQ-hybrid HPKE. Figure 4 indicated that RSA is much more expensive that HPKE. Table II shows the cost of key generation, encryption and decryption for the maximum plaintext size encrypted with RSAES-OAEP for RSA2048, RSA3072 and RSA4096. RSA key generation can be considered an offline step. It also shows the key generation, KEM encapsulation and decapsulation and symmetric encryption/decryption for X25519, `X25519_Kyber`, `X25519_SIKE` HPKE for the same size small plaintexts. We can observe that the symmetric primitive cost is negligible for small plaintexts. Thus, without loss of accuracy we can just compare RSA encryption/decryption with HPKE key generation and encapsulation/decapsulation. RSAES-OAEP with RSA2048 is $\sim$4.8 and $\sim$3.1 times more expensive than classical and Kyber-based PQ-hybrid HPKE respectively. RSA3072 and RSA4096 show as $\sim$14.4 and $\sim$21.2 times slower compared to PQ-hybrid `X25519_Kyber` HPKE. Unfortunately that is not the case for `X25519_SIKE` which performs similar to RSAES-OAEP with RSA4096.

Finally, we evaluated HPKE using a different AEAD scheme, ChaCha20-Poly1305. Table III shows the performance breakdown for all classical, PQ-only and PQ-hybrid HPKE variants for both symmetric AEAD schemes (AES-GCM and ChaCha20-Poly1305) and various plaintext sizes. We see that the cost of the asymmetric primitives remains unchanged. AES256-GCM provides better performance than ChaCha20-Poly1305 independently of the length of the encrypted plaintext. The reason is AES-GCM was hardware optimized for our testbed where ChaCha20-Poly1305 was running in software. Regardless of the symmetric encryption performance, both perform efficiently and offer good post-quantum asymmetric encryption options for relatively well-performing PQ KEMs.

| Func | Pt [B] | RSAES-OAEP 2048 | RSAES-OAEP 3072 | RSAES-OAEP 4096 | HPKE X25519 | HPKE X_Kyber | HPKE X_SIKE |
|---|---|---|---|---|---|---|---|
| KG | - | 88,963 | 412,527 | 2,433,58 | 50.79 | 99.17 | 5,278.63 |
| S | - | - | - | - | 211.16 | 318.24 | 8,709.3 |
| R | - | - | - | - | 159.61 | 244.71 | 9,296.5 |
| s | 214 | 51 | - | - | 0.69 | 0.83 | 0.88 |
| o | 214 | 1,715 | - | - | 0.65 | 0.79 | 0.8 |
| s | 342 | - | 108 | - | 0.85 | 0.96 | 1.04 |
| o | 342 | - | 5,232 | - | 0.76 | 0.93 | 0.89 |
| s | 470 | - | - | 180 | 0.94 | 1.10 | 1.18 |
| o | 470 | - | - | 11,763 | 0.90 | 1.08 | 1.04 |

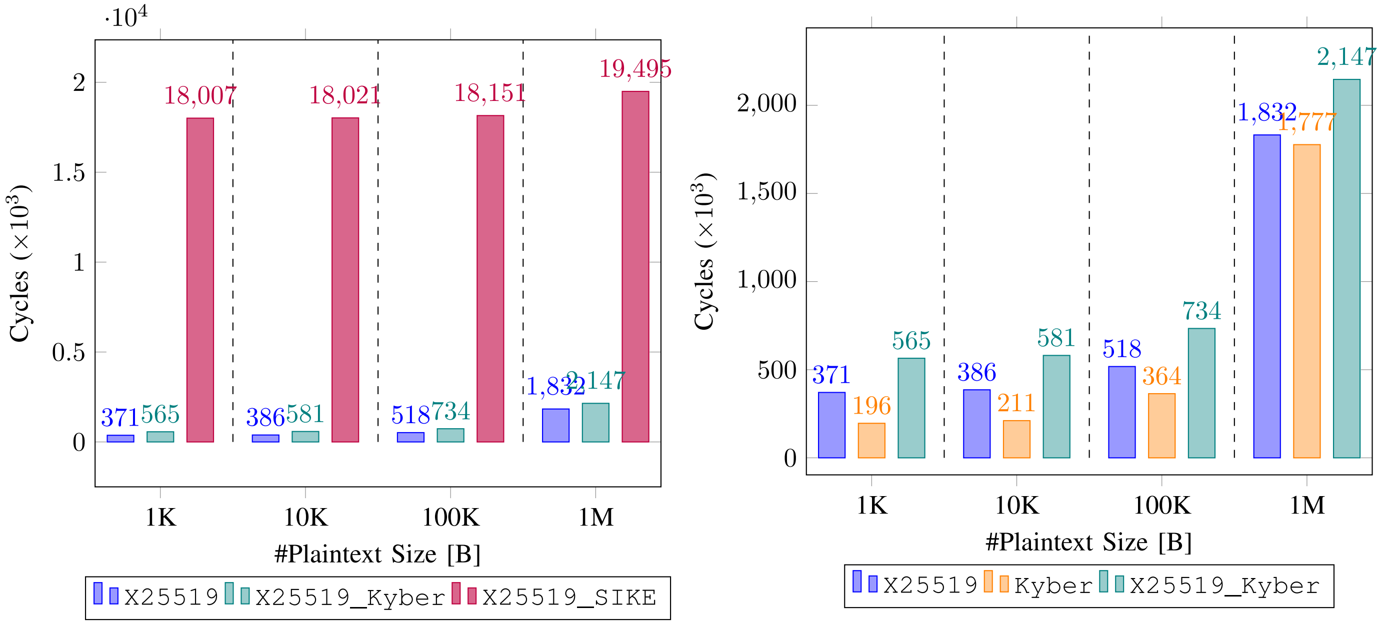
TABLE II: RSA vs PQ-hybrid HPKE performance (Cycles $\times 10^3$) for small plaintexts.

| | Func | Pt [B] | HPKE X25519 | HPKE Kyber | HPKE X25519_Kyber | HPKE SIKE | HPKE X25519_SIKE |
|---|---|---|---|---|---|---|---|
| | KG | - | 50.79 | 40.05 | 99.17 | 5,225.65 | 5,278.63 |
| | S | - | 211.16 | 103.52 | 318.24 | 8,522.93 | 8,709.3 |
| | R | - | 159.61 | 89.18 | 244.71 | 9,161.50 | 9,296.5 |
| AES-256-GCM | s | 1K | 1.34 | 1.53 | 1.52 | 1.54 | 1.54 |
| AES-256-GCM | o | 1K | 1.29 | 1.52 | 1.52 | 1.42 | 1.42 |
| AES-256-GCM | s | 10K | 7.84 | 9.13 | 9.19 | 8.07 | 8.09 |
| AES-256-GCM | o | 10K | 7.77 | 9.08 | 9.08 | 7.87 | 7.88 |
| AES-256-GCM | s | 100K | 73.80 | 85.68 | 85.72 | 73.98 | 74.08 |
| AES-256-GCM | o | 100K | 73.76 | 85.67 | 85.69 | 73.79 | 73.83 |
| AES-256-GCM | s | 1M | 731.21 | 851.29 | 850.98 | 732.81 | 733.10 |
| AES-256-GCM | o | 1M | 730.47 | 733.18 | 732.98 | 733.02 | 732.93 |
| CHACHA20-POLY1305 | s | 1K | 2.26 | 2.53 | 2.53 | 7.79 | 7.77 |
| CHACHA20-POLY1305 | o | 1K | 2.01 | 2.34 | 2.34 | 7.32 | 7.31 |
| CHACHA20-POLY1305 | s | 10K | 37.64 | 17.53 | 17.53 | 48.03 | 47.37 |
| CHACHA20-POLY1305 | o | 10K | 36.65 | 17.07 | 17.06 | 14.78 | 14.78 |
| CHACHA20-POLY1305 | s | 100K | 143.20 | 168.14 | 167.27 | 173.12 | 171.01 |
| CHACHA20-POLY1305 | o | 100K | 140.15 | 162.97 | 162.68 | 140.41 | 140.21 |
| CHACHA20-POLY1305 | s | 1M | 1434.64 | 1589.02 | 1592.08 | 1431.65 | 1434.94 |
| CHACHA20-POLY1305 | o | 1M | 1395.54 | 1402.47 | 1592.08 | 1396.75 | 1395.62 |

TABLE III: Classical, PQ-only, PQ-hybrid HPKE performance (Cycles$\times 10^3$) for two AEADs and various plaintext sizes.

## VII. Conclusion and Future Work

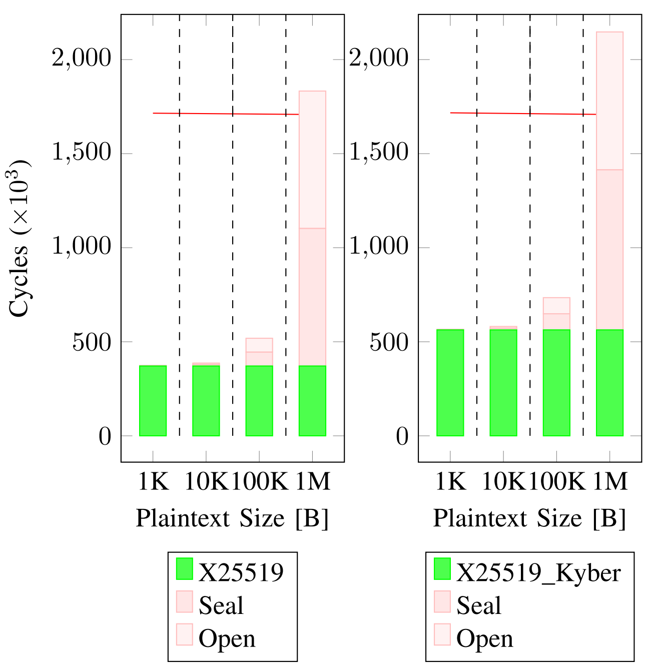
In this work we presented, to the best of our knowledge, the first implementation and performance evaluation of post-quantum HPKE. We extended the scheme to support PQ-only and PQ-hybrid options and integrated it with two PQ KEM algorithms from Round 3 of NIST's PQ Project. We compared these options with RSA and classical HPKE and showed that if the PQ KEM performance is good, the overall scheme is




(a) Classical vs. PQ-hybrid (with Kyber and SIKE)

(b) Classical vs. PQ-hybrid (with Kyber)

Fig. 3: Classical vs PQ HPKE Performance.




(a) Classical HPKE

(b) Hybrid HPKE (Kyber)

Fig. 4: Performance Breakdown for Classical and PQ-hybrid HPKE.

not affected. Additionally the PQ KEM performance overhead is amortized as the size of the plaintext increases.

Future areas of research which could build on this work are PQ HPKE security proofs as described in IV-C and integration with PQ Signatures for authentication.


## Acknowledgements

We would like to thank Adam Petcher for his help with PQ HPKE security proof gaps.


## References


[1] M. Abdalla, M. Bellare, and P. Rogaway. Dhaes: An encryption scheme based on the diffie-hellman problem. *IACR Cryptol. ePrint Arch.*, 1999:7, 1999.

[2] M. Abdalla, M. Bellare, and P. Rogaway. The oracle diffie-hellman assumptions and an analysis of dhies. In *Cryptographers' Track at the RSA Conference*, pages 143–158. Springer, 2001.

[3] G. Alagic, G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, et al. *Status report on the first round of the NIST post-quantum cryptography standardization process*. US Department of Commerce, National Institute of Standards and Technology ..., 2019.

[4] J. Alwen, B. Blanchet, E. Hauck, E. Kiltz, B. Lipp, and D. Riepel. Analysing the HPKE standard. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–116. Springer, 2021.

[5] X. ANSI. 63: Elliptic curve key agreement and key transport protocols. *Working Draft, Oct*, 1998.

[6] AWS. AWS-LC Library, Mar. 2022. https://github.com/awslabs/aws-lc.

[7] E. Barker and L. Chen. Recommendation for Key-Derivation Methods in Key-Establishment Schemes, 2020. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr2.pdf.

[8] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-12, Internet Engineering Task Force, Oct. 2021. Work in Progress.

[9] R. Barnes, K. Bhargavan, B. Lipp, and C. A. Wood. Hybrid Public Key Encryption. Internet-Draft draft-irtf-cfrg-hpke-12, Internet Engineering Task Force, Sept. 2021. Work in Progress.

[10] R. Barnes, K. Bhargavan, B. Lipp, and C. A. Wood. Hybrid Public Key Encryption. RFC 9180, Feb. 2022.

[11] M. Bellare and P. Rogaway. Minimizing the use of random oracles in authenticated encryption schemes. In *International Conference on Information and Communications Security*, pages 1–16. Springer, 1997.

[12] K. Bhatele, A. Sinhal, and M. Pathak. A novel approach to the design of a new hybrid security protocol architecture. In *2012 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, pages 429–433. IEEE, 2012.

[13] N. Bindel, J. Brendel, M. Fischlin, B. Goncalves, and D. Stebila. Hybrid key encapsulation mechanisms and authenticated key exchange. Cryptology ePrint Archive, Report 2018/903, 2018. https://ia.cr/2018/903.

[14] M. Campagna and E. Crockett. Hybrid post-quantum key encapsulation methods (pq kem) for transport layer security 1.2 (tls). *Internet Engineering Task Force, Internet-Draft draft-campagna-tls-bike-sike-hybrid*, 1, 2019.

[15] M. Campagna and A. Petcher. Security of hybrid key encapsulation. *Cryptology ePrint Archive*, 2020.

[16] H. Cheng, J. Großschädl, J. Tian, P. B. Rønne, and P. Y. Ryan. High-throughput elliptic curve cryptography using avx2 vector instructions. In *International Conference on Selected Areas in Cryptography*, pages 698–719. Springer, 2020.

[17] Cloudflare. Encrypting your WAF Payloads with Hybrid Public Key Encryption (HPKE), Feb. 2022. https://blog.cloudflare.com/encrypt-waf-payloads-hpke/.

[18] Cloudflare. Using HPKE to Encrypt Request Payloads, Feb. 2022. https://blog.cloudflare.com/using-hpke-to-encrypt-request-payloads/.

[19] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[20] E. Crockett, C. Paquin, and D. Stebila. Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh. *Cryptology ePrint Archive*, 2019.

[21] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[22] A. Faz-Hernández, J. López, and R. Dahab. High-performance implementation of elliptic curve cryptography using vector instructions. *ACM Transactions on Mathematical Software (TOMS)*, 45(3):1–35, 2019.

[23] H. Fujii and D. F. Aranha. Curve25519 for the cortex-m4 and beyond. In *International Conference on Cryptology and Information Security in Latin America*, pages 109–127. Springer, 2017.

[24] V. Gayoso Martínez, F. Hernández Álvarez, L. Hernández Encinas, and C. Sánchez Ávila. Analysis of ecies and other cryptosystems based on elliptic curves. 2011.

[25] V. Gayoso Martínez, L. Hernández Encinas, and A. Queiruga Dios. Security and practical considerations when implementing the elliptic curve integrated encryption scheme. *Cryptologia*, 39(3):244–269, 2015.

[26] V. Gayoso Martínez, L. Hernández Encinas, and C. Sánchez Ávila. A survey of the elliptic curve integrated encryption scheme. 2010.

[27] V. Gayoso Martínez, F. Hernández Á lvarez, L. Hernández Encinas, and C. Sánchez Á vila. A comparison of the standardized versions of ecies. In *2010 Sixth International Conference on Information Assurance and Security*, pages 1–4, 2010.

[28] T. Geoghegan, C. Patton, E. Rescorla, and C. A. Wood. Privacy Preserving Measurement. Internet-Draft draft-gpew-priv-ppm-00, Internet Engineering Task Force, Oct. 2021. Work in Progress.

[29] S. Gueron and V. Krasnov. Fast prime field elliptic-curve cryptography with 256-bit primes. *Journal of Cryptographic Engineering*, 5(2):141–151, 2015.

[30] M. Hamburg. Ed448-goldilocks, a new elliptic curve. *IACR Cryptol. ePrint Arch.*, 2015:625, 2015.

[31] J. Herranz, D. Hofheinz, and E. Kiltz. Some (in) sufficient conditions for secure hybrid encryption. *Information and Computation*, 208(11):1243–1257, 2010.

[32] F. Idrizi, F. Dalipi, and E. Rustemi. Analyzing the speed of combined cryptographic algorithms with secret and public key. *International Journal of Engineering Research and Development*, 8(2):45, 2013.

[33] E. Jintcharadze and M. Iavich. Hybrid implementation of twofish, aes, elgamal and rsa cryptosystems. In *2020 IEEE East-West Design & Test Symposium (EWDTS)*, pages 1–5. IEEE, 2020.

[34] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, Feb. 2003.

[35] B. S. Kaliski. IEEE p1363: A standard for RSA, Diffie-Hellman, and Elliptic-Curve cryptography. In *International Workshop on Security Protocols*, pages 117–118. Springer, 1996.

[36] C. F. Kerry and P. D. Gallagher. Digital signature standard (dss). *FIPS PUB*, pages 186–4, 2013.

[37] A. Langley, M. Hamburg, and S. Turner. Elliptic Curves for Security. RFC 7748, Jan. 2016.

[38] B. Lipp. An analysis of hybrid public key encryption, 2020.

[39] V. G. Martínez, L. H. Encinas, et al. A comparison of the standardized versions of ecies. In *2010 Sixth International Conference on Information Assurance and Security*, pages 1–4. IEEE, 2010.

[40] D. Moody, G. Alagic, D. C. Apon, D. A. Cooper, Q. H. Dang, J. M. Kelsey, Y.-K. Liu, C. A. Miller, R. C. Peralta, R. A. Perlner, et al. Status report on the second round of the nist post-quantum cryptography standardization process. 2020.
[41] NIST. Migration to Post-Quantum Cryptography.
[42] NIST. NIST PQ project, Feb. 2022. https://csrc.nist.gov/projects/post-quantum-cryptography.
[43] C. Paquin, D. Stebila, and G. Tamvada. Benchmarking post-quantum cryptography in tls. In *International Conference on Post-Quantum Cryptography*, pages 72–91. Springer, 2020.
[44] F. I. P. S. PUBLICATION. Digital Signature Standard (DSS), Feb. 2022. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.
[45] E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631, June 1999.
[46] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, Aug. 2018.
[47] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood. TLS Encrypted Client Hello. Internet-Draft draft-ietf-tls-esni-13, Internet Engineering Task Force, Aug. 2021. Work in Progress.
[48] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
[49] H. Seo. Memory efficient implementation of modular multiplication for 32-bit arm cortex-m4. *Applied Sciences*, 10(4):1539, 2020.
[50] H. Seo and R. Azarderakhsh. Curve448 on 32-bit arm cortex-m4. In *International Conference on Information Security and Cryptology*, pages 125–139. Springer, 2020.
[51] H. Seo, Z. Liu, Y. Nogami, T. Park, J. Choi, L. Zhou, and H. Kim. Faster ecc over \mathbb {F} _ {2^{521}-1} (feat. neon). In *ICISC 2015*, pages 169–181. Springer, 2015.
[52] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.
[53] V. Shoup. Information technology-Security techniques-Encryption algorithms-Part 2: Asymmetric Ciphers. *ISO/IEC 18033-2*, 2004.
[54] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis. Assessing the overhead of post-quantum cryptography in tls 1.3 and ssh. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 149–156, 2020.
[55] S. Singanamalla, S. Chunhapanya, J. Hoyland, M. Vavruša, T. Verma, P. Wu, M. Fayed, K. Heimerl, N. Sullivan, and C. Wood. Oblivious dns over https (odoh): A practical privacy enhancement to dns. *Proceedings on Privacy Enhancing Technologies*, 2021(4):575–592, 2021.
[56] D. Stebila, S. Fluhrer, and S. Gueron. Hybrid key exchange in TLS 1.3. Internet-Draft draft-ietf-tls-hybrid-design-04, Internet Engineering Task Force, Jan. 2022. Work in Progress.
[57] D. Stebila and J. Green. Elliptic Curve Algorithm Integration in the Secure Shell Transport Layer. RFC 5656, Dec. 2009.
[58] The BouncyCastle Team. BouncyCastle crypto lib, Feb. 2022. https://www.bouncycastle.org.
[59] The Cryptopp Team. Cryptopp crypto lib, Feb. 2022. https://cryptopp.com.
[60] The NaCl Team. NaCl crypto library, Jan. 2022. https://nacl.cr.yp.to.
[61] C. Tjhai, M. Tomlinson, G. Bartlett, S. Fluhrer, D. V. Geest, O. Garcia-Morchon, and V. Smyslov. Multiple Key Exchanges in IKEv2. Internet-Draft draft-ietf-ipsecme-ikev2-multiple-ke-04, Internet Engineering Task Force, Sept. 2021. Work in Progress.