A formal analysis of OpenPGP's post-quantum public-key algorithm extension

Duong Dinh Tran[1,*], Kazuhiro Ogata[1] and Santiago Escobar[2]

[1]Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan
[2]VRAIN, Universitat Politècnica de València, Valencia, Spain

Abstract

OpenPGP is an open standard widely used for encrypting and decrypting communication data. Because the security of the current public-key algorithms is dangered by the rapid advancement of quantum computers, a post-quantum extension of the OpenPGP protocol has been proposed in which traditional public-key primitives based on elliptic curve cryptography (ECC) are used in combination with CRYSTALS-Kyber and CRYSTALS-Dilithium, two presumed quantum-resistant schemes. This paper presents a formal analysis of the OpenPGP's post-quantum extension with Maude, a formal specification language and a high-performance tool. We model the protocol in Maude as a state machine with the presence of two honest participants together with an active attacker who can intercept network connections and specifically, break the ECC-based public-key algorithms. The security properties under analysis are (1) *secrecy of messages* and (2) *authenticity of messages*. By using the Maude search functionality, we verify that the protocol enjoys (1) and (2) up to depths 13 and 11, respectively.

Keywords

OpenPGP, post-quantum, formal analysis, security verification

1. Introduction

OpenPGP [1, 2] is an open standard of the Pretty Good Privacy (PGP) security program, a popular system used for encrypting and decrypting communications, preventing unwanted individuals from eavesdropping. The most widely used application of the OpenPGP protocol is to allow individuals to exchange emails securely by encrypting and decrypting their contents. Besides, OpenPGP also supports protecting private texts, files, and even whole disk partitions. OpenPGP supports many public-key encryption and digital signature algorithms. Initially, the RSA algorithm was used as the public-key encryption scheme. To protect a message, the sender encrypts it with a random session key that is then encrypted by the receiver's RSA public key. Only the recipient can decrypt the session key by using the RSA private key and uses it to decrypt the encrypted message. Over time, to meet the security requirement, the RSA key size was increased, which made it lose efficiency. Elliptic Curve Cryptography (ECC), on the other hand, can provide the same security level as RSA with a smaller key size. The OpenPGP standard was then extended to support signing and key exchange based on ECC by RFC 6637 [3]

FAVPQC 2023: International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, November 21, 2023, Brisbane, Australia.
*Corresponding author.
duongtd@jaist.ac.jp (D. D. Tran); ogata@jaist.ac.jp (K. Ogata); sescobar@upv.es (S. Escobar)

in 2012. Instead of using the recipient's RSA public key and private key to encrypt and decrypt the session key, the sender and recipient use a shared secret established by the Elliptic Curve Diffie-Hellman (ECDH) method to symmetrically encrypt the session key.

The security of the currently used public-key algorithms, including RSA, ECDH, and ECC-based digital signatures, is threatened by the rapid advancement of quantum computers recently. On a sufficiently large quantum computer, Shor's algorithm [4] allows attackers to break those public-key algorithms by efficiently computing the private key from the public key. Facing that threat, in 2017, The National Institute of Standards and Technology (NIST) started the Post-Quantum Cryptography Standardization Project [5], which aims to standardize a new class of public-key algorithms that are resistant to quantum computers as measures before practical quantum computers become operational. A post-quantum extension for the OpenPGP protocol [6] has been proposed and standardized by an Internet Engineering Task Force (IETF) working group. In this extension, a traditional ECC-based public-key encryption scheme is used in combination with CRYSTALS-Kyber [7, 8], a Key Encapsulation Mechanism (KEM), and an ECC-based digital signature is used in combination with CRYSTALS-Dilithium [9]. CRYSTALS-Kyber and CRYSTALS-Dilithium base their security on the hardness of the *learning with errors* problem, which is presumed to be hard even with quantum computers. They are two of the first group of winners in the standardization project as NIST announced in July 2022[1]. Let us shortly refer to them as Kyber and Dilithium. The OpenPGP's post-quantum extension is referred to as PQ OpenPGP in this paper.

In this paper, we present a formal analysis of PQ OpenPGP. We use Maude [10], a formal specification language and a high-performance tool to model the protocol as a state machine. We employ the equipped invariant model checker in Maude (the **`search`** command) to check two properties: (1) *secrecy of messages* and (2) *authenticity of messages*. The former guarantees that messages are securely exchanged, that is, no one other than the sender and the recipient can learn the message content. The latter states that if Bob successfully decrypts an encrypted message apparently sent from Alice and verifies the signature, obtaining a raw message M, then Alice indeed sent M. To model the presence of an attacker, we rely on the standard Dolev-Yao model [11]. There exists a generic attacker who can control all connections in the network. Particularly, the attacker can intercept any message sent in the network to collect information from such a message, use all cryptographic primitives to manipulate the available information, and pretend some individual to send a faking message to another individual. Besides, our specification also allows the attacker to break the security of the ECDH method and the ECC-based digital signature schemes, that is, the attacker can derive the associated private key from a given public key.

The Maude specification of the protocol is publicly available on the webpage[2]. The remainder of this paper is organized as follows. Some closely related work is mentioned in Section 2. Next, in Section 3, we briefly introduce some preliminaries, which are necessary for the rest of the paper. OpenPGP and its post-quantum extension are described in Section 4. Then, Section 5 and Section 6 present how to model the protocol in Maude and the formal analysis. Finally, Section 7 summarizes our study.

[1] https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
[2] https://github.com/duongtd23/pq-openpgp-analysis

2. Related work

There are several case studies on applications of formal methods to security analysis of post-quantum cryptographic protocols. S. Gazdag et al. [12] have conducted a security analysis of a post-quantum version of the Internet Key Exchange version 2 (IKEv2). IKEv2 is used to set up a security association (SA) in the Internet Protocol Security (IPsec) protocol [13], the most popular technology for setting up Virtual Private Networks (VPNs). They showed that in a quantum setting, i.e., an attacker has access to quantum computers, the original IKEv2 is insecure, which is similar to what we showed that Eve can learn the ECDH shared secret. They analyzed an IKEv2's post-quantum extension proposed in [14, 15], showing that the improved protocol enjoys all desired security properties. The formal method tool Tamarin [16] was used for the verification, where Tamarin is well known as one of the most useful tools for formal analysis of cryptographic protocols. The Tamarin specification of the protocol is essentially a state machine in which each state is an AC-collection of *facts* and transitions between states, which model the protocol execution, the behavior of honest participants, and capabilities of the attacker, are specified by Tamarin *rules*. Tamarin *facts* and *rules* are closely similar to observable components (i.e, name-value pairs) and Maude rewrite rules, respectively. To check a property, the tool performs an exhaustive, symbolic search based on constraint solving until either a satisfying trace is found or no more rewrite rules can be applied. Because symbolic search is executed, Tamarin can give security proof with respect to an unbounded number of sessions and protocol participants. Meanwhile, with our analysis reported in this paper, the number of participants must be fixed.

A. Hülsing et al. [17] have proposed a WireGuard's post-quantum extension and presented a formal verification of the desired security properties inherited from WireGuard [18], a VPN protocol focusing on simplicity, fast speed, and high performance. The tool Tamarin [16] was also used for the formal verification. They modeled the primitives, messages, etc., used in the protocol as function symbols and terms, which is similar to our approach with Maude. In addition, the paper also presented a computational security proof, which gave stronger security guarantees than the symbolic proof with Tamarin since probability and complexity are taken into account and fewer idealizing assumptions are made. However, more security properties are verified in the verification with Tamarin, and more importantly, it is computer-verified.

There are some existing case studies on analyzing cryptographic protocols and primitives also by using Maude [19, 20]. In this paper, to verify the security properties, we use only the Maude search command (or invariant model checker). Whereas, liveness and fairness properties were taken into account in the analysis experiments reported in [20]. They employed the LTL model checker in Maude. On the other hand, they modeled the protocol under analysis at a higher level than us, that is, some more simplifications are made in their model.

The Amazon Web Services has proposed a quantum-resistant version of the Transport Layer Security 1.2 protocol [21], called the Hybrid Post-Quantum TLS [22], and its formal analysis has been studied in [23]. The analysis was conducted by using Maude-NPA [24] and Par-Maude-NPA [25], a parallel version of Maude-NPA. Maude-NPA is a cryptographic protocol analysis tool implemented in Maude, based on narrowing & rewriting logic [26]. They used *strands* [27] to model the execution of the Hybrid Post-Quantum TLS protocol as well as the capabilities of a Dolev-Yao attacker. Two properties are verified: (1) secrecy property of the post-quantum

key encapsulation mechanism's shared secret, and (2) authentication property. To check each property, a Maude-NPA attack state is defined, representing the insecure state that violates the property. For the analysis, different from Maude's forward search, Maude-NPA performs a backward search from the attack state to check whether it is reachable from an initial state. The backward reachability analysis is symbolic, and hence, Maude-NPA can provide security proof with respect to an unbounded number of sessions and participants, which is an advantage over our analysis approach with Maude. Because of the very large state space generated, similar to our results reported in this paper, they can only guarantee that the protocol enjoys (1) and (2) up to depths 12 and 18, respectively, after about 72 days.

3. Preliminaries

This section gives the definition of key encapsulation mechanisms followed by a brief description of the Maude language.

Key Encapsulation Mechanism

According to the NIST standardization project, key exchange algorithms are formulated as Key Encapsulation Mechanisms (KEMs). The following is the definition of KEMs.

Definition 1. A key encapsulation mechanism is a tuple of algorithms (`KeyGen`, `Encaps`, `Decaps`) along with a finite key space $\mathcal{K}$:

- `KeyGen`() $\rightarrow (pk, sk)$: A probabilistic *key generation* algorithm that outputs a public key pk and a private key sk.
- `Encaps`(pk) $\rightarrow (c, k)$: A probabilistic *encapsulation* algorithm that takes as input a public key pk, and outputs a ciphertext (or encapsulation) c and a shared secret $k \in \mathcal{K}$.
- `Decaps`(c, sk) $\rightarrow k$: A deterministic *decapsulation* algorithm that takes as inputs a ciphertext c and a private key sk, and outputs a shared secret $k \in \mathcal{K}$.

Maude

Maude is a declarative language and high-performance tool that focuses on simplicity, expressiveness, and performance to support the formal specification and analysis of concurrent programs/systems in rewriting logic. In Maude, the most basic building block is *modules*. There are two kinds of modules in Maude: functional modules and system modules. A functional modules is in the form of **`fmod`** $\mathcal{M}$ **`is`** (Σ, E) **`endfm`**, where $\mathcal{M}$ is the module name, Σ is an order-sorted signature and E is an equation set. Σ may contain declarations of sorts (or types), subsorts, operators (or function symbols), and variables. A system module is in the form of **`mod`** $\mathcal{R}$ **`is`** (Σ, E, R) **`endm`**, where R is a set of rewrite rules. Using rewrite rules, we can specify state transitions of a state machine. A rewrite rule starts with the keyword **`rl`**, followed by a label enclosed with square brackets and a colon, two patterns connected with =>, and ends with a full stop. A conditional one starts with the keyword **`crl`** and has a condition following the keyword **`if`** before a full stop. The following is a form of a conditional rewrite rule:

crl [lb] : l => r **if** ... /\ c_i /\ ... .

where lb is a label and c_i is part of the condition, which may be an equation $lc_i = rc_i$. The negation of $lc_i = rc_i$ could be written as $(lc_i =/= rc_i) = \texttt{true}$, where `= true` could be omitted. If the condition ... /\ c_i /\ ... holds under some substitution σ, $\sigma(l)$ can be replaced with $\sigma(r)$.

Maude provides the search command that can find a state reachable from t such that the state matches p and satisfies condition(s) c:

search [n,m] in MOD : t =>* p such that c .

where `MOD` is the name of the module specifying the state machine, and `n` and `m` are optional arguments stating a bound on the number of desired solutions and the maximum depth of the search, respectively. `n` typically is 1 and t typically represents an initial state of the state machine.

4. OpenPGP's post-quantum extension

We describe the OpenPGP protocol followed by its post-quantum extension.

OpenPGP and ECC in OpenPGP

The process that A (the sender) sends a message M to B (the recipient) is shown in Figure 1, which consists of the following sequence of steps:

1. The sender hashes the message (using the hash function H), and then signs the digest with his/her private key (sk_A).
2. The sender randomly selects a key value k, which is served as a *session key* to encrypt the message. Note that this session key is used for this message only, i.e., different messages use different session keys. For the analysis, we suppose that the session key is unique and non-guessable.
3. The sender asymmetrically encrypts (aenc) the session key with the recipient's public key (pk_B), obtaining C_1.
4. The sender concatenates the message's signature and the message, and symmetrically encrypts (senc) it under the session key k, obtaining C_2 ($||$ denotes the concatenation). The concatenation of C_1 and C_2 are outputted as the result.

The process shown in Figure 1 provides confidentiality and integrity to the message being sent. Depending on the sender's desire, it may not be desirable to have both security services. For instance, when integrity is not desirable, the sender will omit the signing part and proceed to encrypt the message only. In this study, however, we always consider both confidentiality and integrity to be desirable.

Figure 2 depicts how the recipient decrypts the received ciphertext and validates it, which consists of the following step sequence:

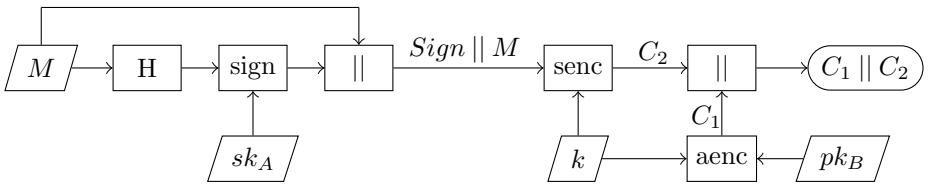

Figure 1: OpenPGP: Sending a message M

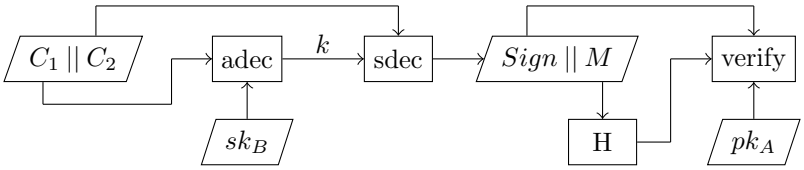

Figure 2: OpenPGP: Receiving an encrypted message

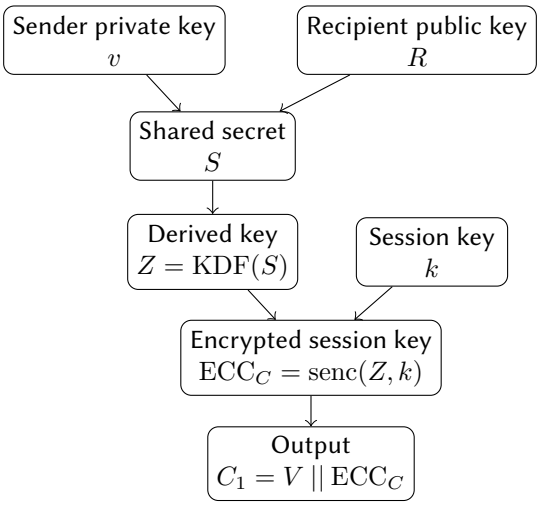

Figure 3: Session key encryption with ECDH

1. The recipient decrypts the first part of the ciphertext (C_1) with his/her private key (sk_B) to recover the session key k.

2. The recipient uses the session key to decrypt the second part of the ciphertext (C_2), obtaining a message and a signature.

3. The recipient hashes the message and uses the sender public key (pk_A) to validate the integrity of the message received.

When the ECDH key exchange method is used, instead of using the recipient's public key and private key to encrypt and decrypt the session key as shown in Figure 1 and Figure 2, a so-called *derived key* is computed from the shared secret and that derived key is used to encrypt

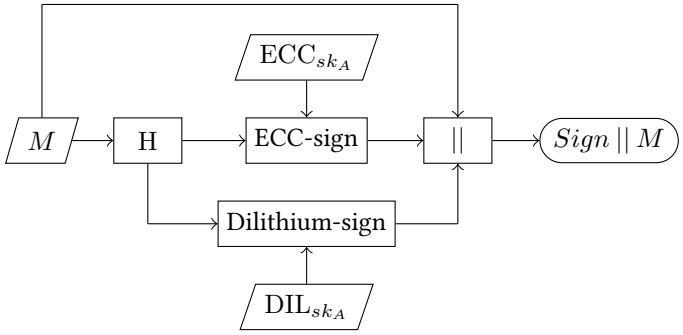

Figure 4: Composite ECC-based signature scheme and CRYSTALS-Dilithium

the session key. The precise step sequence to encrypt the session key with ECDH is given in Figure 3:

1. The sender first generates an ECDH ephemeral private/public key pair $\{v, V\}$.
2. The ECDH shared secret S is computed from v and the recipient public key R.
3. The sender then uses the key derivation function (KDF) to compute the derived key Z from the shared secret.
4. The session key k is encrypted under the derived key.
5. Finally, the sender concatenates his/her ephemeral public key and the encrypted session key, outputting the result.

Upon receiving the message, the recipient computes the shared secret from his/her ECDH private key (the associated private key of R) and the sender public key V. The recipient then derives Z, uses it to decrypt the session key, and uses the session key to decrypt the message as well as validate the signature.

OpenPGP's post-quantum extension

PQ OpenPGP makes use of two pairs of combinations: (1) an ECC-based signature algorithm in combination with CRYSTALS-Dilithium, and (2) ECDH in combination with CRYSTALS-Kyber. Figure 4 illustrates the composite ECC-based and Dilithium signature schemes. Given a message M, the sender hashes it, and signs the message digest with both ECC-based's and Dilithium's sign functions. Note that ECC$_{sk_A}$ and DIL$_{sk_A}$ denote the ECC-based and Dilithium signing private key of the sender, respectively. When decrypting the message, the recipient must validate both signatures.

The Internet-Draft [6] also defines another option in which SPHINCS+ [28] is used as a standalone public-key signature scheme. That option, however, is omitted in this paper.

Figure 5 depicts the composite algorithm with ECDH and Kyber to encrypt a session key k, which consists of the following sequence of steps:

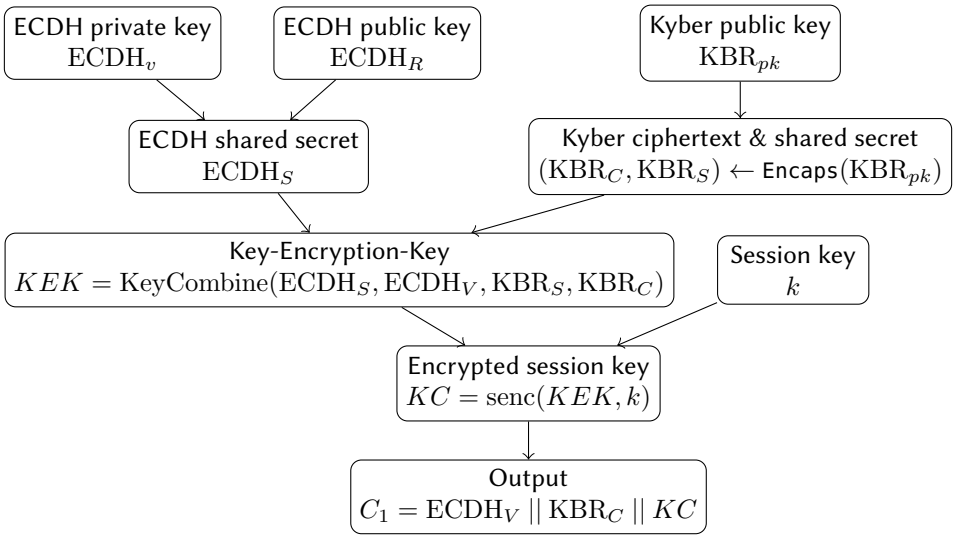

Figure 5: Composite ECDH and CRYSTALS-Kyber

1. The sender first generates an ECDH ephemeral private/public key pair $\{\mathrm{ECDH}_v, \mathrm{ECDH}_V\}$.
2. The ECDH shared secret ECDH_S is computed from the sender's private key and the recipient's public key ECDH_R.
3. Taking the Kyber public key of the recipient as input, the sender performs Kyber `Encaps` to generate a ciphertext KBR_C and a shared secret KBR_S. Note that because `Encaps` is probabilistic, each time it is invoked, different ciphertext and shared secret will be generated.
4. The sender then uses a so-called function $\mathrm{KeyCombine}$ to compute the so-called Key-Encryption-Key KEK from the two shared secrets, the sender's ECDH public key, and the Kyber ciphertext.
5. The session key k is encrypted under the Key-Encryption-Key.
6. Finally, the sender concatenates his/her ECDH public key, the Kyber ciphertext, and the encrypted session key, outputting the result.

Upon receiving the message, the recipient computes: (1) the ECDH shared secret from his/her private key (the associated private key of R) and the sender public key V; and (2) the Kyber shared secret by performing `Decaps`, taking his/her private key (the associated private key of KBR_{pk}) and the received Kyber ciphertext as inputs. The recipient then derives Key-Encryption-Key, uses it to decrypt the session key, and uses the session key to decrypt the message as well as validate the two signatures.

For the analysis, we suppose that the binding between a public key and its owner identifier is always correctly done. It means that the attacker is unable to claim their public key is a public

key of some honest participant. To achieve this purpose, different methods are used together in OpenPGP, such as, public-key fingerprints, certificates, and web of trust. Web of trust is a decentralized trust model, in contrast to the centralized trust model that relies on a certificate authority, allowing anyone to sign a partner public key to certify that that public key truly belongs to that person. Everyone can choose their trusted introducers, from which they trust all certificates signed by such introducers. In fact, however, no completely satisfactory solution (including the web of trust model) is known to the issue of binding between a public key and its owner.

5. Modeling the protocol

This section presents how to model PQ OpenPGP in Maude [10], a formal specification language and a high-performance tool that can be used to analyze a wide range of systems/protocols.

5.1. Modeling ECDH and CRYSTALS-Kyber

ECDH is modeled in the following Maude functional module:

```
fmod ECDH is
  sorts EdPubKey EdPriKey EdShareS .
      derivation of the associated public key from a private key
  op pk  : EdPriKey            -> EdPubKey .
      computation of the shared secret from a received public key and a private key
  op ss  : EdPubKey EdPriKey   -> EdShareS .
      constructor of a shared secret is a private key pair
  op  |  : EdPriKey EdPriKey   -> EdShareS [comm] .
  vars SK SK2 : EdPriKey .            declarations of variables
  eq ss(pk(SK), SK2) = SK | SK2 .     an equation
endfm
```

This module first introduces three sorts (or types) `EdPubKey`, `EdPriKey`, and `EdShareS` representing ECDH public keys, private keys, and shared secrets, respectively. We then declare three operators (or function symbols), where the comments followed by give their corresponding explanations for what they stand for. For instance, the first operator takes a private key as input and returns its associated public key as output. The equational attribute `comm` attached to the last operator denotes that it is commutative, namely, $(sk_1 \mid sk_2)$ and $(sk_2 \mid sk_1)$ are identical with all private keys sk_1 and sk_2. The equation defines the semantic of the last operator.

We model Kyber based on the general definition of KEMs, i.e., Definition 1. Another module is introduced, which contains the following declarations and definitions:

```
sorts KbPubKey KbPriKey KbShareS KbCipher .
    KeyGen is a probabilistic algorithm,
    so keygen takes a private key as input and returns the public key
op keygen  : KbPriKey          -> KbPubKey .
    similarly, Encaps is probabilistic, so an argument of KbPriKey is added
op encapsC : KbPubKey KbPriKey -> KbCipher .       Encaps: returns ciphertext
op encapsK : KbPubKey KbPriKey -> KbShareS .       Encaps: returns shared secret
op decaps  : KbCipher KbPriKey -> KbShareS .       Decaps
```

```
    constructor of a shared secret is a private key pair
op  &      : KbPriKey KbPriKey -> KbShareS .
vars SK SK2 : KbPriKey .
eq encapsK(keygen(SK), SK2) = (SK & SK2) .
eq decaps(encapsC(keygen(SK), SK2), SK) = (SK & SK2) .
```

We introduce four sorts `KbPubKey`, `KbPriKey`, `KbShareS`, and `KbCipher`, representing public keys, private keys, shared secrets, and ciphertexts (or encapsulations), respectively. As indicated in Definition 1, `KeyGen` and `Encaps` are two probabilistic algorithms. Thus, to specify them as deterministic procedures in Maude, an argument of the sort `KbPriKey` is added as the input argument. With `Encaps`, we introduce two separate operators `encapsC` and `encapsK`, respectively returning the ciphertext and the shared secret. The first equation defines the semantic of the operator `&`. Given a ciphertext and a private key, the second equation states that `Decaps` properly outputs the shared secret only if the given ciphertext encapsulates the public key that is the associated public key of the given private key.

5.2. Modeling cryptographic primitives

We define the generic sort `Data` as the supersort of all sorts mentioned before, such as `EdPubKey` and `KbPubKey`. `||` is the concatenation operator, which forms terms of the sort `DataL`.

```
sorts Data DataL .
subsorts EdPubKey EdPriKey KbPubKey KbPriKey ... < Data .      some other sorts in ...
subsort Data < DataL .
op nilDL :             -> DataL [ctor] .
op  ||   : DataL DataL -> DataL [assoc ctor id: nilDL] .       concatenation
```

`nilDL` is the identity element of `||`, thanks to the Maude attribute `id:`. `assoc` states that `||` is associative. `ctor` states that `nilDL` and `||` are constructors of the sort `DataL`.

Several operators are introduced to model cryptographic primitives. For instance, the operators modeling the hash function, the sign and verify functions of ECC-based digital signature schemes are as follows:

```
op h        : DataL               -> Data .          hash function
op ecSign   : EsPriKey Data       -> Data .          ECC based signature: sign
                      sign digest
op ecVerify : EsPubKey Data Data  -> Bool .          ECC based signature: verify
```

where `EsPriKey` and `EsPubKey` are two sorts of the signing private keys and the verifying public keys. Given a digest, a verifying public key, and a signature, the following two equations define that `ecVerify` returns valid only if the signature is signed with the associated private key of the given public key (with the attached attribute `owise`, the second equation will not be applied unless the first equation cannot be applied):

```
eq ecVerify(pkes(SKES), ecSign(SKES,D), D) = true .
eq ecVerify(PKES,       SIGN,           D) = false [owise] .
```

5.3. Modeling the protocol execution

We suppose that there exists two honest users together with a dishonest user (the intruder) participating in the protocol. PQ OpenPGP is modeled as a state machine, where each state is in the form of a braced associative-commutative collection (AC-collection) of name-value pairs, which are called observable components in Maude.

We use the following observable components:

- (`ecdh`$[a]$: `< pk(`$sked$`) ; `$sked$` >`): User a has an ECDH public/private key pair `pk(`$sked$`)` and $sked$, where `< • ; • >` is defined as a pair of keys. Recall that the operator `pk` takes a private key as input and returns its associated public key as output. In fact, a user may have more than one ECDH public/private key pair, but for the sake of simplicity, in this analysis, we restrict each user to only one ECDH public/private key pair.

- (`ecsig`$[a]$: `< pkes(`$skes$`) ; `$skes$` >`): User a has an ECC-based signature scheme's public/private key pair `pkes(`$skes$`)` and $skes$, where `pk(`$skes$`)` denotes the associated public key of $skes$.

- (`kyber`$[a]$: `< keygen(`$skkb$`) ; `$skkb$` >`): User a has a Kyber public/private key pair `keygen(`$skkb$`)` and $skkb$.

- (`dilit`$[a]$: `< pkdi(`$skdi$`) ; `$skdi$` >`): User a has a Dilithium signature public/private key pair `pkdi(`$skdi$`)` and $skdi$, where `pkdi(`$skdi$`)` is the associated public key of $skes$.

- (`rd-sesskey`: ks): ks is a space-separated list of available session keys to be used by any user. That is, whenever a user needs to randomly generate a session key to encrypt a raw message, the head value of ks will be used. Note that we define ks as a list instead of a set to reduce the number of states generated in the checking experiments later on.

- (`e-knl`: ds): The intruder's knowledge is ds, which is a set of `Data`'s terms separated by `;`. ds includes all information that is available to the intruder, such as those grasped from messages exchanged between other users, and those synthesized by using cryptographic primitives.

- (`nw`: ems): The network, i.e., the collection of messages exchanged, is ems. Each encrypted message is in the form of `msg(`$a, b, cipher$`)`, where a, b, and $cipher$ respectively are the sender, the recipient, and the message content.

- (`ms`: ms): ms is a space-separated list of raw messages to be sent by users.

- (`rd-ecdh`: $skeds$): $skeds$ is a list of available values to be used as ECDH ephemeral private keys. That is, whenever a user queries a new ECDH ephemeral private key, the head value of $skeds$ will be used.

- (`rd-kyber`: $skkbs$): $skkbs$ is a list of available values to be used as Kyber ephemeral private keys.

- (`used-kyber`$[a]$: $skkbs$): $skkbs$ is a set of Kyber ephemeral private keys that was used by user a and was erased from a's memory.

The initial state is defined as follows:

```
eq init =
 {(ecsig[a]: (< pkes(skes1) ; skes1 >))
  (dilit[a]: (< pkdi(skdi1) ; skdi1 >))
  (ecdh[a] : (< pk(sked1) ; sked1 >))     (ecdh[b] : (< pk(sked2) ; sked2 >))
  (kyber[b]: (< keygen(skkb1) ; skkb1 >)) (kyber[a]: (< keygen(skkb2) ; skkb2 >))
  (ecsig[eve]: (< pkes(skes3) ; skes3 >)) (dilit[eve]: (< pkdi(skdi3) ; skdi3 >))
  (ecdh[eve] : (< pk(sked3) ; sked3 >))   (kyber[eve]: (< keygen(skkb3) ; skkb3 >))
  (rd-sesskey: (k1 k2))  (rd-ecdh: sked4) (rd-kyber: skkb4)
  (e-knl: (pkes(skes1) ; pkdi(skdi1) ; pk(sked1) ; pk(sked2) ;
           keygen(skkb1) ; pkes(skes3) ; skes3 ; pkdi(skdi3) ; skdi3 ;
           pk(sked3) ; sked3 ; keygen(skkb3) ; skkb3))
  (nw: empty)  (ms: (m1 m2))
  (used-kyber[b]: empty) (used-kyber[a]: empty)} .
```

a and b are the two honest users, while eve is the dishonest one. We suppose that b always be the recipient, while a always be the sender, and so two key pairs of ECC-based and Dilithium signatures are given to a. Initially, the intruder's knowledge includes their own keys and all other participants' public keys.

Encryption and sending a message is modeled by the following rewrite rule:

```
crl [send] :
   {(ms: (M MS)) (rd-sesskey: (K KS))
    (ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
    (ecdh[B] : (< PKED ; SKED > , SKS3)) (ecdh[A] : (< PKED2 ; SKED2 > , SKS4))
    (kyber[B]: (< PKKB ; SKKB > , SKS5)) (kyber[A] : (< PKKB2 ; SKKB2 > , SKS6))
    (nw: NW) (e-knl: DS) (used-kyber[A]: SKS7) OCs}
=> {(ms: MS)     (rd-sesskey: KS)
    (ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
    (ecdh[B] : (< PKED ; SKED > , SKS3)) (ecdh[A] : (SKS4))
    (kyber[B]: (< PKKB ; SKKB > , SKS5)) (kyber[A] : (SKS6))
    (nw: (msg(A,B, PKED2 || KBC || KC || C2) NW))
    (e-knl: (DS ; PKED2 ; KBC ; KC ; C2))
    (used-kyber[A]: (SKS7 , < PKKB2 ; SKKB2 >)) OCs}
if H    := h(M) /\
   SIGN := ecSign(SKES,H)  /\  SIGN2 := diSign(SKDI,H) /\
   EDSS := ss(PKED,SKED2)  /\  KBSS := encapsK(PKKB,SKKB2) /\
   KBC  := encapsC(PKKB,SKKB2) /\
   KEK  := kcombine(EDSS,PKED2,KBSS,KBC) /\
   KC   := senc(KEK, K) /\
   C2   := senc(K, SIGN || SIGN2 || M) .
```

M is the raw message being sent. Sender A first hashes M, obtaining H, and then signs H with his own ECC-based signature's and Dilithium's private keys, obtaining SIGN and SIGN2, respectively. PKED and PKKB are the ECDH's and Kyber's public keys of recipient B, while SKED2 and SKKB2 are the ECDH's and Kyber's ephemeral private keys of A. From these keys, A computes the two shared secrets EDSS and KBSS as well as the Kyber ciphertext KBC. The key-encryption-key KEK is then computed by the function kCombine (KeyCombine in Figure 5). A selects session key K, encrypts it with the key-encryption-key, obtaining KC, and encrypts SIGN || SIGN2 || M with

K, obtaining C2. The encrypted message msg(A,B, PKED2 || KBC || KC || C2) is put into the network in the successor state, namely, A sent that message to B. At the same time, the intruder learns the message content (the message content is put into e-knl), A erases his ECDH's and Kyber's ephemeral key pairs just used, and the Kyber's ephemeral private key is put into A's set of Kyber's ephemeral private keys used.

We suppose that the sender always deletes ECDH's and Kyber's ephemeral keys after sending a message. In contrast, the recipient is not forced to do so. Typically, users are required to periodically update the ephemeral keys. We defined two other rewrite rules modeling the process by which users generate new ephemeral key pairs.

5.4. Modeling the intruder

The intruder is given a capability of intercepting any message sent in the network to learn information carried in that message. This capability has been illustrated through the rewrite rule [send] previously presented. In addition, the intruder can (1) generate random session keys and raw messages to be sent, and (2) apply any cryptographic primitive function to derive new information from the information is available to them. Several rewrite rules are defined to model (2), some among them are as follows:

```
    hash anything
rl [hash] :
  {(e-knl: (M ; DS)) OCs} => {(e-knl: (M ; DS ; h(M))) OCs} .
    sign anything, SKES is any ECC based signature scheme's private key
rl [sign] :
   {(e-knl: (SKES ; h(D) ; DS)) OCs}
=> {(e-knl: (SKES ; h(D) ; DS ; ecSign(SKES,h(D)))) OCs} .
    compute keys and encapsulation by Encaps
    PKKB and SKKB are variables of Kyber public and private keys
crl [encaps] :
   {(e-knl: (PKKB ; SKKB ; DS)) OCs}
=> {(e-knl: (PKKB ; SKKB ; DS ; encapsC(PKKB, SKKB) ; encapsK(PKKB, SKKB))) OCs}
if PKKB =/= keygen(SKKB) .
```

With the rewrite rule [hash], the intruder is restricted to hash only raw messages instead of any terms of sort Data. Even though this restricts the intruder capabilities to some extent, it helps to reduce the number of states generated in the analysis later on. Recall that because we define ds in (e-knl: ds) as a set, there is no duplication in ds. As a result, it cannot be the case that an infinite number of different states are produced by applying the rule [hash] infinite times with the same message M.

We assume practical quantum computers are available to the intruder, from which they can break the security of ECDH and ECC-based signature schemes. If a public key of those two schemes is given to Eve, Eve can derive its associated private key. This capability is specified by the following two rewrite rules:

```
    breaking ECDH, Eve can derive private keys from public keys
rl [break-ecdh] :
   {(e-knl: (pk(SKED) ; DS)) OCs}   => {(e-knl: (pk(SKED) ; DS ; SKED)) OCs} .
    breaking ECC based signature schemes
```

```
rl [break-ecc-sign] :
   {(e-knl: (pkes(SKES) ; DS)) OCs} => {(e-knl: (pkes(SKES) ; DS ; SKES)) OCs} .
```

6. Formal analysis

We first check that the formal specification allows two users to successfully send an encrypted message and decrypt it. To this end, we define a Maude **search** command finding a state reachable from `init` such that an encrypted message sent from `A` to `B` exists in the network and `B` successfully decrypts and validates the message. Executing that **search** command, Maude almost immediately found such a state at the first depth, that is, only one rewrite rule (`send`) is applied, changing the state from `init` to the state found.

6.1. Secrecy of messages

All messages should be securely sent, that is, no one other than the sender and the recipient can learn the message content. This property can be called the *secrecy of messages*. To check this property, we define the following **search** command:

```
search [1,10] in PQOPENPGP : init =>*
 {(ecsig[A]: (< PKES ; SKES > , SKS))  (dilit[A]: (< PKDI ; SKDI > , SKS2))
  (ecdh[B] : (< PKED ; SKED > , SKS3)) (kyber[B]: (< PKKB ; SKKB > , SKS5))
  (nw: (msg(A,B, PKED2 || KBC || KC || C2) NW))
  (e-knl: (M ; DS)) OCs}
such that
  (A =/= eve and B =/= eve)   /\
  EDSS := ss(PKED2, SKED)     /\  KBSS := decaps(KBC, SKKB) /\
  KEK  := kcombine(EDSS, PKED2, KBSS, KBC) /\
  K    := sdec(KEK, KC) /\
  SIGN || SIGN2 || M := sdec(K, C2) /\
  ecVerify(PKES, SIGN,  h(M)) /\  diVerify(PKDI, SIGN2, h(M)) .
```

The command tries to find a state with bounded depth 10 in which there exists an encrypted message sent from `A` to `B` in the network, `B` successfully decrypts the raw message `M` and verifies the composite signatures, and `M` exists in Eve's knowledge. The procedure for the recipient `B` to decrypt and validate the message is as follows. `B` first computes (1) the ECDH shared secret `EDSS` from the received public key and her own private key; and (2) the Kyber shared secret `KBSS` by the function `Decaps` with the received encapsulation and her own private key. `B` then computes the key-encryption-key `KEK` and uses it to decrypt the session key `K`. `B` uses the session key to decrypt the encrypted message, obtaining a raw message `M` and two signatures. Finally, `B` verifies the two signatures.

Maude did not find such a state after about 1 minute and 39 seconds, meaning that the protocol enjoys the property up to depth 10. The experiment was conducted on a computing server that carries 192 GB of memory and a 2.8 GHz microprocessor with 8 cores. The experimental results are reported in Table 1 and will be discussed in detail below.

Property	Depth	Result	Time (h:m:s)	No. States
Secrecy of messages	8	∅	0:00:6.7	46317
	9	∅	0:00:22.2	98943
	10	∅	0:01:39	206972
	11	∅	0:08:31	430750
	12	∅	0:42:34	903344
	13	∅	5:08:16	1929731
Authenticity of messages	8	∅	0:06:40	46317
	9	∅	0:23:39	98943
	10	∅	1:26:30	206972
	11	∅	6:54:14	430750

Table 1
Experimental results. ∅ means that Maude did not find solution(s) for the given search command.

6.2. Learning ECDH shared secrets

We define another command to show that Eve can learn the ECDH shared secret after a message is exchanged between two users. This **search** command is identical to the previous command used to check the secrecy property of messages sent except that the observable component `e-knl` is now changed to the following:

```
(e-knl: (EDSS ; DS))
```

Recall that `EDSS` is the ECDH shared secret. Maude found a state after about 6 milliseconds. The found state is at depth 3 after the three following rewrite rules are applied:

1. rule `[send]`: `a` encrypts and sends a message to `b`.
2. rule `[break-ecdh]`: `eve` derives the ECDH ephemeral private key of `a` from the public key.
3. rule `[ss]`: `eve` computes the ECDH shared secret from `a`'s private key just obtained and `b`'s public key.

6.3. Authenticity of messages

This property states that if Bob successfully decrypts an encrypted message apparently sent from Alice and successfully verifies the composite signatures with Alice's verifying keys, obtaining a raw message M, then Alice indeed sent M to Bob. To check the property, we define the following **search** command:

```
search [1,10] in PQOPENPGP : init =>*
 {(ecsig[A]: (< PKES ; SKES > , SKS))   (dilit[A]: (< PKDI ; SKDI > , SKS2))
  (ecdh[B] : (< PKED ; SKED > , SKS3))  (kyber[B]: (< PKKB ; SKKB > , SKS5))
  (e-knl: (PKED2 ; KBC ; KC ; C2 ; DS)) (nw: NW)
  (used-kyber[A]: (< PKKB2 ; SKKB2 > , SKS6)) OCs}
such that
  (A =/= eve and B =/= eve)   /\
```

```
EDSS := ss(PKED2, SKED)     /\  KBSS := decaps(KBC, SKKB) /\
KEK  := kcombine(EDSS, PKED2, KBSS, KBC) /\
K    := sdec(KEK, KC) /\
SIGN || SIGN2 || M := sdec(K, C2) /\
ecVerify(PKES, SIGN,  h(M)) /\  diVerify(PKDI, SIGN2, h(M)) /\
not(msg(A,B, PKED2 || KBC || KC || C2) \in NW) /\
KBC == encapsC(PKKB, SKKB2) .
```

The command tries to find a state with bounded depth 10 such that:

(1) Eve knows four pieces of data `PKED2`, `KBC`, `KC`, and `C2` (Eve can use these pieces of data forming a message, and pretend `A` to send the message to `B`);

(2) from those four pieces of data, `B` successfully decrypts a raw message `M` and verifies the signatures with `A`'s verifying keys;

(3) `A` did not send those four pieces of data to `B`, namely, such a message does not exist in the network.

If such a state exists, it is possible to happen a case in which `B` receives a valid message apparently sent from `A`, but the message is actually forged by Eve. Maude did not find such a state after about 1 hour and 26 minutes. The **`search`** command bounds `KBC` to be the ciphertext encapsulating an ephemeral private key of `A`, meaning that the ciphertext was really created by `A`. As mentioned before, we suppose that the binding between a public key (or KEM ciphertext) and its real owner is always correctly done.

Table 1 shows the experimental results of (1) the *secrecy of messages* from depth 8 to depth 13 and (2) the *authenticity of messages* from depth 8 to depth 11. Those results confirm that PQ OpenPGP enjoys (1) and (2) up to depths 13 and 11, respectively. With the same depth, checking (1) is significantly faster than checking (2), which is mostly because, in each state, there is a huge number of substitutions for the following pattern in the **`search`** command of (2):

```
(e-knl: (PKED2 ; KBC ; KC ; C2 ; DS))
```

`KC` and `C2` are variables of the sort `Data`, and so they can be substituted by any terms of `Data` or its subsorts. Besides, `;` is AC, making the number of substitution solutions increase.

7. Concluding remarks

Using the Maude tool, we have presented a formal analysis of the OpenPGP's post-quantum extension. The protocol is modeled as a state machine with three participants, one of whom is an active attacker. The experimental results have confirmed that the protocol enjoys two properties: *secrecy of messages* and *authenticity of messages* up to some specific depths. Even though we limit the number of protocol participants to three, the number of the state space generated is really huge (nearly 1 million states at depth 12), which makes us unable to proceed with the experiments at deeper depths due to time limitations. State explosion, however, is not dedicated to Maude only, but rather the burdensome issue of all analysis/verification approaches based on model checking.

Interactive theorem proving, the complementary approach to model checking, is one possible way to escape from the state explosion issue. Protocol analysis/verification based on interactive theorem proving makes it feasible to deal with an infinite state space. In particular, with cryptographic protocols, we can do formal verification with an unbounded number of participants and session executions. However, this comes at a cost: human creativity is required. Verification by using the tool IPSG [29] to produce the so-called proof scores [30] is a promising way that we plan to apply the method to the OpenPGP's post-quantum extension.

References

[1] P. Wouters, D. Huigens, J. Winter, N. Yutaka, OpenPGP, Internet-Draft draft-ietf-openpgp-crypto-refresh-10, Internet Engineering Task Force, 2023. URL: https://datatracker.ietf.org/doc/draft-ietf-openpgp-crypto-refresh/10/, work in Progress.

[2] H. Finney, L. Donnerhacke, J. Callas, R. L. Thayer, D. Shaw, OpenPGP Message Format, RFC 4880, 2007. doi:10.17487/RFC4880.

[3] A. Jivsov, Elliptic Curve Cryptography (ECC) in OpenPGP, RFC 6637, 2012. doi:10.17487/RFC6637.

[4] P. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in: Proceedings 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124–134. doi:10.1109/SFCS.1994.365700.

[5] National Institute of Standards and Technology (NIST), Post-quantum cryptography, NIST, 2017. URL: https://csrc.nist.gov/projects/post-quantum-cryptography.

[6] S. Kousidis, F. Strenzke, A. Wussler, Post-Quantum Cryptography in OpenPGP, Internet-Draft draft-wussler-openpgp-pqc-02, Internet Engineering Task Force, 2023. URL: https://datatracker.ietf.org/doc/draft-wussler-openpgp-pqc/02/, work in Progress.

[7] J. W. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber: A CCA-secure module-lattice-based KEM, in: 2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018, IEEE, 2018, pp. 353–367. doi:10.1109/EuroSP.2018.00032.

[8] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Kyber: Algorithm Specifications And Supporting Documentation (version 3.02), 2021. URL: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[9] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé, CRYSTALS-Dilithium – Algorithm Specifications and Supporting Documentation (Version 3.1), 2021. URL: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[10] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, C. L. Talcott, Programming and symbolic computation in maude, J. Log. Algebraic Methods Program. 110 (2020). doi:10.1016/j.jlamp.2019.100497.

[11] D. Dolev, A. C. Yao, On the security of public key protocols, IEEE Trans. Inf. Theory 29 (1983) 198–207.

[12] S. Gazdag, S. Grundner-Culemann, T. Guggemos, T. Heider, D. Loebenberger, A formal analysis of IKEv2's post-quantum extension, in: ACSAC '21: Annual Computer Security

Applications Conference, Virtual Event, USA, December 6 - 10, 2021, ACM, 2021, pp. 91–105. doi:10.1145/3485832.3485885.

[13] S. Frankel, S. Krishnan, IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap, RFC 6071, 2011. doi:10.17487/RFC6071.

[14] V. Smyslov, Intermediate Exchange in the Internet Key Exchange Protocol Version 2 (IKEv2), RFC 9242, 2022. doi:10.17487/RFC9242.

[15] C. Tjhai, M. Tomlinson, G. Bartlett, S. Fluhrer, D. V. Geest, O. Garcia-Morchon, V. Smyslov, Multiple Key Exchanges in the Internet Key Exchange Protocol Version 2 (IKEv2), RFC 9370, 2023. doi:10.17487/RFC9370.

[16] D. A. Basin, C. Cremers, J. Dreier, R. Sasse, Symbolically analyzing security protocols using TAMARIN, ACM SIGLOG News 4 (2017) 19–30. doi:10.1145/3157831.3157835.

[17] A. Hülsing, K. Ning, P. Schwabe, F. Weber, P. R. Zimmermann, Post-quantum wireguard, in: 2021 IEEE Symposium on Security and Privacy, 2021, pp. 304–321. doi:10.1109/SP40001.2021.00030.

[18] J. A. Donenfeld, Wireguard: Next generation kernel network tunnel, in: 24th Annual Network and Distributed System Security Symposium, NDSS 2017, 2017. https://www.wireguard.com/papers/wireguard.pdf.

[19] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani, Kyber, Saber, and SK-MLWR Lattice-Based Key Encapsulation Mechanisms Model Checking with Maude, IET Information Security 2023 (2023) 9399887. doi:10.1049/2023/9399887.

[20] V. García, S. Escobar, K. Ogata, S. Akleylek, A. Otmani, Modelling and verification of post-quantum key encapsulation mechanisms using maude, PeerJ Comput. Sci. 9 (2023) e1547. doi:10.7717/PEERJ-CS.1547.

[21] E. Rescorla, T. Dierks, The Transport Layer Security (TLS) Protocol Version 1.2, RFC 5246, 2008. doi:10.17487/RFC5246.

[22] M. Campagna, E. Crockett, Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS), RFC, RFC Editor, 2021. URL: https://datatracker.ietf.org/doc/html/draft-campagna-tls-bike-sike-hybrid.

[23] D. D. Tran, C. M. Do, S. Escobar, K. Ogata, Hybrid Post-Quantum Transport Layer Security formal analysis in Maude-NPA and its parallel version, PeerJ Comput. Sci. (2023). To appear.

[24] S. Escobar, C. Meadows, J. Meseguer, A Rewriting-Based Inference System for the NRL Protocol Analyzer and Its Meta-Logical Properties, Theor. Comput. Sci. 367 (2006) 162–202. doi:10.1016/j.tcs.2006.08.035.

[25] C. M. Do, A. Riesco, S. Escobar, K. Ogata, Parallel Maude-NPA for cryptographic protocol analysis, in: K. Bae (Ed.), Rewriting Logic and Its Applications - 14th International Workshop, WRLA@ETAPS 2022, Munich, Germany, April 2-3, 2022, Revised Selected Papers, volume 13252 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 253–273. doi:10.1007/978-3-031-12441-9 13.

[26] S. Escobar, J. Meseguer, P. Thati, Narrowing and rewriting logic: from foundations to applications, in: F. J. López-Fraguas (Ed.), Proceedings of the 15th Workshop on Functional and (Constraint) Logic Programming, WFLP 2006, Madrid, Spain, November 16-17, 2006, volume 177 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2006, pp. 5–33. doi:10.1016/j.entcs.2007.01.004.

[27] F. J. Thayer, J. C. Herzog, J. D. Guttman, Strand spaces: Why is a security protocol correct?, in: Security and Privacy - 1998 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 3-6, 1998, Proceedings, IEEE Computer Society, 1998, pp. 160–171. doi:10.1109/SECPRI.1998.674832.

[28] J.-P. Aumasson, D. J. Bernstein, W. Beullens, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, B. Westerbaan, SPHINCS+ – Submission to the 3rd round of the NIST post-quantum project. v3.1, 2022. URL: https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

[29] D. D. Tran, K. Ogata, Formal verification of TLS 1.2 by automatically generating proof scores, Computers & Security 123 (2022) 102909. doi:https://doi.org/10.1016/j.cose.2022.102909.

[30] K. Ogata, K. Futatsugi, Proof scores in the OTS/CafeOBJ method, in: Formal Methods for Open Object-Based Distributed Systems, 6th IFIP WG 6.1 International Conference, FMOODS 2003, Paris, France, volume 2884 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 170–184. doi:10.1007/978-3-540-39958-2\ 12.