IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 10 October 2023, accepted 16 December 2023, date of publication 28 December 2023,
date of current version 5 January 2024.

Digital Object Identifier 10.1109/ACCESS.2023.3347914

RESEARCH ARTICLE

Formal Analysis of Post-Quantum Hybrid Key Exchange SSH Transport Layer Protocol

DUONG DINH TRAN[1], KAZUHIRO OGATA[1], SANTIAGO ESCOBAR[2], SEDAT AKLEYLEK[3,4], AND AYOUB OTMANI[5]

[1]Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923-1211, Japan
[2]VRAIN, Universitat Politècnica de València, 46022 Valencia, Spain
[3]Ondokuz Mayıs University, 55139 Samsun, Turkey
[4]University of Tartu, 50090 Tartu, Estonia
[5]University of Rouen Normandie, 76821 Rouen, France

Corresponding author: Duong Dinh Tran (duongtd@jaist.ac.jp)

The work of Duong Dinh Tran and Kazuhiro Ogata was supported by the JST SICORP, Japan, under Grant JPMJSC20C2. The work of Santiago Escobar was supported in part by the MCIN/AEI/10.13039/501100011033 and ERDF—A way of making Europe under Grant PID2021-122830OB-C42, in part by the Generalitat Valenciana under Grant CIPROM/2022/6, in part by the MICIN/AEI/10.13039/501100011033 under Grant PCI2020-120708-2, and in part by the European Union NextGenerationEU/PRTR. The work of Sedat Akleylek was supported in part by TUBITAK under Grant 121R006. The work of Ayoub Otmani was supported in part by the FAVPQC Project funded by CNRS, and in part by the Agence Nationale de la Recherche (ANR) within France 2030 Program under Grant ANR-22-PETQ-0008 PQ-TLS.

ABSTRACT Facing the quantum attack threat, a quantum-resistant version of the SSH Transport Layer protocol has been proposed and been standardized by an IETF working group. This standardization process has been motivated by the fact that if practical quantum computers become available, classical key exchange algorithms used today will no longer be safe because their security can be efficiently broken by Shor's algorithm running on a quantum computer. In this paper, we construct a symbolic model of the proposed protocol, specify it in the specification language CafeOBJ, and conduct a formal analysis of the claimed security properties with the employment of a formal method tool called Invariant Proof Score Generator (IPSG). Three security properties are formally verified with respect to an unbounded number of protocol participants and protocol executions by employing IPSG to produce their formal proofs, the so-called *proof scores* in CafeOBJ. With another property, namely *authentication*, we find a counterexample against it, and then we propose to slightly revise the protocol. With the improved version, we formally verify that the *authentication* property is enjoyed, while the three properties remain secure. To model the presence of malicious participants, we extend the Dolev-Yao intruder, which is the standard intruder model used in security protocol analysis/verification, because the availability assumption of large-scale quantum computers gives the attacker some new capabilities. Under our threat model, the intruder is given capabilities of fully controlling the network as the Dolev-Yao intruder, and additionally breaking ECDH's security as well as compromising secrets.

INDEX TERMS Formal verification, post-quantum SSH, CafeOBJ, proof score, security analysis.

I. INTRODUCTION

The development of quantum computing has caused solid danger to public-key encryption systems widely used today. This comes from the ability of quantum-based algorithms, such as Shor's algorithm [1], in breaking the presumed difficulty of some mathematical problems on which those public-key encryption systems are based. Although right now there is no quantum computer with enough power to break the real cryptosystems currently used, with a huge research and development investment recently from many tech giants, such as Intel, IBM, and Google, large-scale quantum computers are promisingly becoming available in the near future. Besides, attackers can record the encrypted information from now and later decrypt it when large-scale quantum computers become available, which is known as the *harvest*

The associate editor coordinating the review of this manuscript and approving it for publication was Junaid Arshad.

now and decrypt later attack [2]. These are motivations for the early construction of cryptographic algorithms that are resistant to quantum attackers, the so-called post-quantum cryptographic algorithms. Facing the quantum attack threat, in 2017, the National Institute of Standards and Technology (NIST) launched a competition to standardize post-quantum public-key cryptographic algorithms.[1]

The Secure Shell protocol (SSH) [3] gives users a secure way to access a computer over an unsecured network. The most well-known application of SSH is to allow users, particularly system administrators, to securely remote login to a server and execute commands through the command line environment. SSH consists of three sub-protocols, among which, the SSH Transport Layer protocol [4] is in charge of key negotiation in order to establish symmetric keys, which are then used by the other protocols [5], [6] to securely exchange information between two participants. Due to the threat of quantum attacks, an Internet Engineering Task Force (IETF) Working Group has proposed a post-quantum version of the SSH Transport Layer protocol [7]. The proposed protocol is being standardized and its latest version is 01 by June 2023. The proposed protocol, which is abbreviated as PQ SSH in this paper, bases its security on the so-called post-quantum hybrid key exchange method that uses a classical key exchange algorithm and a quantum-resistant Key Encapsulation Mechanism (KEM) in parallel. In the IETF Draft [7], the former is fixed to Elliptic Curve Diffie-Hellman (ECDH), while CRYSTALS-Kyber [8] is chosen for the latter.

In this paper, we present a security analysis of PQ SSH version 01 [7] by using two formal method tools CafeOBJ [9] and IPSG [10]. CafeOBJ [9] is an advanced algebraic language for writing formal specifications of a wide variety of systems, supporting order-sorted equational logic with various equational theory attributes such as associative, commutativity, identity, and idempotency. It is equipped with a rich syntax and many useful features, such as module expressions, modules instantiated parameters using views, and the flexible mix-fix syntax so that it can be used to formally specify even complex systems like concurrent and distributed systems. Besides, CafeOBJ can be used as a powerful interactive theorem proving system, where humans can write a formal proof, the so-called proof score [11], [12], for an invariant property under verification. That proof score is executable, and the formal verification is done by executing it with CafeOBJ. IPSG (Invariant Proof Score Generator) could automate the proof score writing process to produce the proof score for an invariant property. Precisely, given a CafeOBJ formal specification, invariant properties formalizing the desired properties we want to verify, and an auxiliary lemma list, IPSG can automatically produce the proof scores verifying those properties.

Analysis and verification of security protocols, such as Transport Layer Security (TLS) [13], [14] and SSH [3], require the assumption of the presence of malicious participants in addition to honest participants, which is an essential difference from verification of other systems/protocols. The Dolev-Yao generic intruder model [15] is used for this purpose as a de facto standard. To tackle security analysis of post-quantum cryptographic protocols, such as PQ SSH [7], however, we need to extend the Dolev-Yao intruder model because the availability assumption of large-scale quantum computers gives the intruder some new capabilities. Many quantum algorithms have been proposed. Among them, Shor's algorithm [1] and Grover's algorithm [16] are considered potential threats to public-key primitives and symmetric primitives, respectively, used today. We can work around Grover's algorithm by doubling the symmetric key length. Thus, Shor's algorithm is the only one for which we need to come up with new public-key cryptographic primitives to make cyberspace in the quantum era secure. To this end, NIST has been then launching the competition to standardize new public-key primitives, such as KEMs. We have formally specified and analyzed/verified some KEMs [17] to comprehend KEMs better and to be able to make an abstract version of them in order to use in the analysis/verification of higher-level protocols, such as PQ SSH in this paper. That study has also helped us to be able to extend the Dolev-Yao generic intruder model so as to conduct the formal analysis reported in this paper.

Our main contributions in this study are summarized as follows:

- A symbolic model of the protocol that is formally specified in CafeOBJ as a specification, faithfully captures what is specified in the IETF Draft [7]. We use an extended version of the Dolev-Yao intruder model [15] to specify a threat model in which the intruder has the capabilities of fully controlling the network, breaking ECDH's security (assumedly by using large-scale quantum computers), and compromising secrets.
- Formal verifications that the protocol enjoys three desired security properties including (1) *session key secrecy*, (2) *forward secrecy*, and (3) *session identifier uniqueness*, where IPSG is used to generate the proof scores. The verifications are achieved with respect to an unbounded number of protocol participants and session executions.
- We consider another property, namely *authentication*, which we find a counterexample against the property. We propose to slightly revise the protocol by adding the identifiers of the client and the server into the exchange hash. We revise the CafeOBJ formal specification accordingly so that we can formally verify that the improved protocol enjoys the *authentication* property as well as (1), (2), and (3).

The webpage[2] provides all materials used in this paper, including the CafeOBJ formal specifications, the proof

[1] https://csrc.nist.gov/projects/post-quantum-cryptography

[2] https://github.com/duongtd23/PQSSH

scores verifying the desired properties, and the installation guidelines for the tools used. The rest of this paper is organized as follows. First, we mention some closely related work and background requirements to understand the rest of the paper in Sections II and III, respectively. Then, Section IV introduces the PQ SSH protocol as well as different approaches to post-quantum cryptographic algorithm construction. Section V presents the formal specification of the protocol in CafeOBJ. The formal analysis of the four properties is reported in Section VI. Finally, we summarize our study in Section VII.

II. RELATED WORK

Symbolic analysis and computation analysis are known as two complementary approaches to security analysis of cryptographic protocols. The computational approach is widely used by cryptographers as a standard way to verify the security of cryptographic protocols, while formal method researchers typically prefer the symbolic approach. Many studies and supporting tools on security analysis of cryptographic protocols in both symbolic and computational approaches have been surveyed in [18] and [19]. In the symbolic approach, to the best of our knowledge, [20] and [21] are the only two studies on post-quantum cryptographic protocol analysis. The former has symbolically analyzed the Ephemeral DH Over COSE (EDHOC) protocol [22], which was proposed by the Lightweight Authenticated Key Exchange Working Group. The protocol is dedicated to being used on IoT devices. The original protocol uses the DH key exchange algorithm, which is not quantum-safe because the discrete logarithm problem of DH will be no longer hard with the presence of practical quantum computers. The protocol was then made post-quantum secure by replacing the DH algorithm with a quantum-resistant KEM. This KEM-based version was also taken into account in their analysis. An interesting point in this work is that they used Sapic$^+$ [23] verification framework so that the protocol formal specification written in π-calculus [24] can be exported into some other security analyzer tools including ProVerif [25] and Tamarin [26]. A formal analysis of SSH with the employment of Sapic$^+$ was also reported in [23], where the secrecy and authentication properties were formally verified.

WireGuard [27] is a VPN protocol focusing on simplicity, fast speed, and high performance. Facing the quantum attack threat, a quantum-resistant version of WireGuard has been proposed [21], the so-called post-quantum WireGuard (PQ-WireGuard). The authors have symbolically verified the desired security properties of the protocol that are inherited from WireGuard. The symbolic proof used Tamarin prover [26]. They first symbolically modeled the primitives, messages, etc. used in the protocol as function symbols and terms. The verification then formalized the security properties as Tamarin lemmas and introduced some auxiliary lemmas as well. In addition to the symbolic verification, the authors also conducted and reported a computational verification. On the one hand, the symbolic proof exposes the superiority to the computational proof in two points: first, it covers more security properties, and second, it is computer-verified. On the other hand, the computational proof gives stronger security assurances because it took probability and complexity into account, and fewer idealizing assumptions were made.

Formal methods with supporting tools have been successfully used to formally verify the security of various cryptographic protocols. Tamarin [26], ProVerif [25], and Maude-NPA [28] are the most well-known tools, among others. Tamarin is a successor of the analyzer Scyther [29]. It operates based on AC-collection rewriting and its verification algorithm is based on constraint solving. Basically, a specification written in Tamarin is a state machine where each state is an AC-collection of *facts*. Transitions between states are defined by *rules*, which specify the protocol execution, the behavior of honest parties as well as the capabilities of the intruder. Roughly speaking, *facts* and *rules* correspond to *observers* and *transitions* of OTSs in our proof approach. A security property is specified as a trace property, then Tamarin checks the satisfiability and/or the validity of the formula formalizing the property under verification. If it is the validity checking, Tamarin first converts the formula to its the negated form in order to perform a satisfiability checking instead. After that, Tamarin performs an exhaustive, symbolic search based on constraint solving until either a satisfying trace is found or no more rewrite rules can be applied. However, the search is not guaranteed to be terminated for every analysis attempt, and when it is the case, the tool allows manual interaction from human users to operate it with the supplementation of some extra lemmas. Thus, on the one hand, both Tamarin and our verification method require manual efforts on conjecturing additional lemmas, while on the other hand, Tamarin's verification algorithm and the simultaneous induction proof method are essentially different.

ProVerif [25] is another automated tool for symbolically analyzing cryptographic protocols. A variant of the π-calculus [19] is used to model the protocol under verification, and then ProVerif translates it to a set of Horn clauses. This Horn clause representation makes some abstractions, which is the cost for the support of verification with respect to an unbounded number of sessions. Given a desired security property, the tool reduces the problem of finding an attack against the property to the derivability of a fact on the Horn clauses representing the protocol execution. If the fact is not derivable from the clauses, the protocol enjoys the property. Otherwise, if an attack derivation is found, the derivation may correspond to a real attack. However, that attack derivation may also not be feasible in reality because of the internal abstractions performed by ProVerif [30]. If that is the case, the tool just returns an ''unknown'' answer.

Maude-NPA [28] is a formal method tool for cryptographic protocol analysis based on narrowing & rewriting logic [31]. The tool is implemented in Maude [32]. The Dolev-Yao

intruder model [15] and the strand model [33] are used to model the intruder's capabilities. In this manner, the intruder is given the capability of fully controlling the network, for example, intercepting & modifying messages and impersonating some protocol participants to send some messages to other participants. For the analysis, Maude-NPA uses a backward narrowing reachability analysis modulo an equational theory. Narrowing [32] is a generalization of term rewriting that allows logical variables in subject terms and replaces pattern matching by unification. The backward narrowing reachability analysis starts from a final insecure state pattern specified by human users that represents insecure states, the so-called attack pattern, to check whether it is reachable from an initial state. Roughly speaking, the attack pattern and the initial state (if found) in Maude-NPA correspond to the negated formula formalizing the validity property and the satisfying trace (if found), respectively, in Tamarin. The exhaustive search has pros as it is fully automated, but it poses cons because the search would take a long time to terminate when the state space is large even though several optimization techniques to reduce the search state space have been developed [28], [34], [35]. In contrast, running time is not a problem with our verification approach presented in this paper since proof score execution normally takes only a short time. Note, however, that our verification approach is not fully automated because manual efforts are spent to construct some additional lemmas.

Even though verification of designs and specifications of cryptographic protocols has significantly contributed to their reliability, some researchers argued that formal verification by using some formal specification languages to specify cryptographic protocols often lacks some aspect of details. They then proposed to verify detailed protocol implementations, such as verifying SSH implementations in [36] and [37] and verifying implementations of different versions of Transport Layer Security (TLS) protocol in [13], [14], [38], [39], and [40]. On the one hand, this verification technique has the benefit of not having to worry about some potentially erroneous details of the protocol implementation code being missed. On the other hand, the verification may be very costly. First, the verifier may take a very long time to terminate or even may not terminate in some circumstances, especially with a large implementation. Second, a large amount of memory may be consumed.

III. PRELIMINARIES

This section briefly describes some necessary preliminaries for the rest of the paper including OTS, the simultaneous induction proof, and the formal verification with CafeOBJ.

A. OBSERVATIONAL TRANSITION SYSTEM (OTS)

Let Υ denote a universal state space and D, possibly with a subscript, such as D_{o1} and D_o, denote a data type.

Definition 1: An OTS $\mathcal{S}$ is a tuple $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ in which:

- $\mathcal{O}$: A finite set of observers. Each *observer* $o : \Upsilon\ D_{o1} \ldots D_{om} \rightarrow D_o$ takes one state (we assume to be always the first argument) and m ($m \geq 0$) data values and returns one data value.
- $\mathcal{I}$: The set of initial states, where $\mathcal{I} \subseteq \Upsilon$.
- $\mathcal{T}$: A finite set of transitions. Each *transition* $t : \Upsilon\ D_{t1} \ldots D_{tn} \rightarrow \Upsilon$ has the *effective condition* $c\text{-}t : \Upsilon\ D_{t1} \ldots D_{tn} \rightarrow$ Bool such that if $c\text{-}t(\upsilon, x_1, \ldots, x_n)$ does not hold, $t(\upsilon, x_1, \ldots, x_n) =_{\mathcal{S}} \upsilon$ for $x_1 \in D_{t1}, \ldots, x_n \in D_{tn}$.

Definition 2: Given an OTS $\mathcal{S}$, the set of *reachable states* $\mathcal{R}_{\mathcal{S}}$ with respect to (wrt) $\mathcal{S}$ are inductively defined:

- Each $\upsilon \in \mathcal{I}$ is reachable wrt $\mathcal{S}$.
- For each $t \in \mathcal{T}$ and each $x_k \in D_{tk}$ for $k = 1, \ldots, n$, $t(\upsilon, x_1, \ldots, x_n)$ is reachable wrt $\mathcal{S}$ if $\upsilon \in \Upsilon$ is reachable wrt $\mathcal{S}$.

Predicates whose types are $\Upsilon\, D_1 \ldots D_n \rightarrow$ Bool are called *state predicates*. A state predicate that holds in all reachable states is called an *i*nvariants. For example, predicate $p : \Upsilon\, D_1 \ldots D_n \rightarrow$ Bool, where $(\forall \upsilon \in \mathcal{R}_{\mathcal{S}})(\forall d_1 \in D_1) \ldots (\forall d_n \in D_n).\ p(\upsilon, d_1, \ldots, d_n)$, is an invariant wrt $\mathcal{S}$.

B. SIMULTANEOUS INDUCTION PROOF

This section describes the simultaneous induction method [11], [12] for invariant proof. Suppose that we want to prove a predicate $p_1\ \Upsilon\ D_{p1} \ldots D_{pl} \rightarrow$ Bool is invariant wrt an OTS $\mathcal{S}$, i.e, $(\forall \upsilon \in \mathcal{R}_{\mathcal{S}})(\forall x_1 \in D_{p1}) \ldots (\forall x_l \in D_{pl}).\ p_1(\upsilon, x_1, \ldots, x_l)$. To shorten the formula for ease of reading, let us use $\mathbf{D}_1$ to denote $D_{p1} \ldots D_{pl}$ and $\mathbf{x}_1$ to denote $x_1, \ldots, x_l$. This is also applied with other notations in the rest of this paper as well, i.e., a bold upper-case (sans serif) letter denotes a list of data types while a bold lower-case (sans serif) letter represents a list of variables. Consequently, what is needed to prove is $(\forall \upsilon \in \mathcal{R}_{\mathcal{S}})(\forall \mathbf{x}_1 \in \mathbf{D}_1).\ p_1(\upsilon, \mathbf{x}_1)$. We prove that by using induction on the argument of states of p_1. For the base case, we need to prove the following:

$$(\forall \upsilon \in \mathcal{I}_{\mathcal{S}})(\forall \mathbf{x}_1 \in \mathbf{D}_1).\ p_1(\upsilon, \mathbf{x}_1) \tag{1}$$

(1) can typically be proved by deduction (or if we use a theorem prover, it can be straightforwardly resolved by the prover). For each induction case t associated with the transition $t : \Upsilon\ \mathbf{D}_t \rightarrow \Upsilon$, what we need to prove is as follows:

$$\begin{aligned}&(\forall \upsilon \in \mathcal{R}_{\mathcal{S}}).\ ((\forall \mathbf{x}_1 \in \mathbf{D}_1)\, p_1(\upsilon, \mathbf{x}_1)\\ &\quad \Rightarrow (\forall \mathbf{y}_t \in \mathbf{D}_t)(\forall \mathbf{x}_1 \in \mathbf{D}_1)\, p_1(t(\upsilon, \mathbf{y}_t), \mathbf{x}_1))\end{aligned} \tag{2}$$

It suffices to prove $p_1(t(\upsilon, \mathbf{y}'_t), \mathbf{x}'_1)$ for an arbitrary state υ and arbitrary values $\mathbf{y}'_t$ and $\mathbf{x}'_1$ of $\mathbf{D}_t$ and $\mathbf{D}_1$, respectively, under the induction hypotheses $(\forall \mathbf{x}_1 \in \mathbf{D}_1)\, p_1(\upsilon, \mathbf{x}_1)$. The induction hypothesis instance $p_1(\upsilon, \mathbf{x}'_1)$ is often used for that proof. We can also use other instances, such $p_1(\upsilon, \mathbf{x}_2)$ and $p_1(\upsilon, \mathbf{x}_3)$. It is, however, typically impossible to prove (2) standalone for non-trivial p_1. Instead, we often prove the conjunction of p_1 and $k - 1$ other predicates, let's say $p_2, \ldots, p_k$, where $p_i : \Upsilon\, \mathbf{D}_i \rightarrow$ Bool for $i = 2, \ldots, k$. That is, we prove $(\forall\, \mathbf{x}_1 \in \mathbf{D}_1) p_1(\upsilon, \mathbf{x}_1) \wedge \ldots \wedge (\forall \mathbf{x}_k \in \mathbf{D}_k) p_k(\upsilon, \mathbf{x}_k)$ is invariant wrt $\mathcal{S}$. Subsequently, with the induction case t

associated with the transition t, (2) is now changed to:

$$(\forall \upsilon \in \mathcal{R}_{\mathcal{S}}).$$
$$((\forall \mathbf{x}_1 \in \mathbf{D}_1)\, p_1(\upsilon, \mathbf{x}_1) \wedge \ldots \wedge (\forall \mathbf{x}_k \in \mathbf{D}_k)\, p_k(\upsilon, \mathbf{x}_k)$$
$$\Rightarrow (\forall \mathbf{y}_t \in \mathbf{D}_t)\, ((\forall \mathbf{x}_1 \in \mathbf{D}_1)\, p_1(t(\upsilon, \mathbf{y}_t), \mathbf{x}_1) \wedge \ldots$$
$$\wedge (\forall \mathbf{x}_k \in \mathbf{D}_k)\, p_k(t(\upsilon, \mathbf{y}_t), \mathbf{x}_k))) \quad (3)$$

It suffices to prove each conjunct $p_i(t(\upsilon, \mathbf{y}'_t), \mathbf{x}'_i)$ of the conclusion part for an arbitrary state υ and arbitrary values $\mathbf{y}'_t$ and $\mathbf{x}'_i$ of $\mathbf{D}_t$ and $\mathbf{D}_i$, respectively, under the induction hypotheses $(\forall \mathbf{x}_1 \in \mathbf{D}_1) p_1(\upsilon, \mathbf{x}_1) \wedge \ldots \wedge (\forall \mathbf{x}_k \in \mathbf{D}_k) p_k(\upsilon, \mathbf{x}_k)$. Typically, it suffices to use only $p_i(\upsilon, \mathbf{x}'_i)$ as the induction hypothesis instance:

$$p_i(\upsilon, \mathbf{x}'_i) \Rightarrow p_i(t(\upsilon, \mathbf{y}'_t), \mathbf{x}'_i) \quad (4)$$

Sometimes, it is necessary to use some more instances, for example:

$$p_j(\upsilon, \mathbf{x}_j) \Rightarrow p_i(\upsilon, \mathbf{x}'_i) \Rightarrow p_i(t(\upsilon, \mathbf{y}'_t), \mathbf{x}'_i) \quad (5)$$

In this case, we can say that the proof of p_i uses p_j as a lemma. From what has been presented, although k predicates depend on each other, we can prove them compositionally by using induction for each predicate, and in the proof of p_i, p_j could be used to strengthen the induction hypothesis. Therefore, the proof method is called simultaneous induction.

C. FORMAL VERIFICATION WITH CafeOBJ AND IPSG

Based on the simultaneous induction proof method, this section illustrates how to do formal verification with CafeOBJ [9] and IPSG [10]. We consider a simplified version of the classical key distribution protocol proposed by Denning and Sacco [41], which is called SDS (Simplified Denning-Sacco) in this paper. Fig. 1 depicts the two messages exchanged in the SDS protocol. The initiator message sent by Alice is referred to as message (i), while the reply message sent by Bob is referred to as message (ii). aenc and senc denote asymmetric encryption and symmetric encryption, respectively. PK_A and SK_A denote the public and private keys of principal A, respectively. K denotes a secret key, which is unguessable, and ; is the concatenation operator. The two messages exchanged between A and B can be explained as follows. A first selects a secret key K, and encrypts it together with the identifier of B under the private key of A, obtaining a ciphertext. The ciphertext is once more encrypted by the public key of B, and then A sends the obtained result to B. When B receives the message from A, B consecutively decrypts the content received twice respectively with his/her private key and the public key of A. If the two decryptions are successful and the final obtained plaintext consists of the receiver's identifier and a key K, then B responds back to A with the identifiers of A and B symmetrically encrypted by the key K in order to prove the possession of the secret key.

1) MODELING THE PROTOCOL

To do verification with CafeOBJ, we first model the protocol in CafeOBJ with the presence of attackers. We introduce

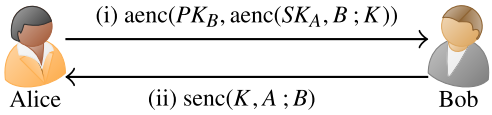

FIGURE 1. Two messages exchanged in the SDS protocol.

CafeOBJ sorts (types) `Prin`, `PriKey`, and `PubKey` to represent protocol participants, their private and public keys, respectively, and three CafeOBJ operators as follows:

```
[Prin PubKey PriKey]
op priK  : Prin -> PriKey {constr}    -- private key
op pubK  : Prin -> PubKey {constr}    -- public key
op intru :      -> Prin   {constr}    -- the intruder
```

The `constr` indicates that, for example, `priK` is a constructor of the sort `PriKey`. The last operator (also called a constant), i.e., `intru`, represents a generic attacker. This attacker is modeled as the Dolev-Yao intruder [15], who can completely control the network. Note that `--` denotes a CafeOBJ comment.

We additionally introduce sort `Secret` representing secret keys, and two generic sorts `Data` and `DataL`, which are the supersorts of `Prin`, `PubKey`, `PriKey`, and `Secret`. `||` is used as the concatenation operator:

```
[Data < DataL]
[Prin PubKey PriKey Secret < Data]
-- concatenation operator
op _||_ : DataL DataL -> DataL {assoc constr}
```

`assoc` indicates that the operator is associative, meaning that with three pieces of data d_1, d_2, and d_3, $(d_1$ `||` $d_2)$ `||` d_3 and d_1 `||` $(d_2$ `||` $d_3)$ are identical.

Symmetric and asymmetric encryptions/decryptions are modeled by the following operators:

```
--        key   plain     ciphertext
op aenc : Data  DataL  -> Data {constr}  -- asymmetric encr
op senc : Data  DataL  -> Data {constr}  -- symmetric encr
--        key   cipher    plaintext
op adec : Data  Data   -> DataL          -- asymmetric decr
op sdec : Data  Data   -> DataL          -- symmetric decr
```

These operators are defined by means of equations, for example:

```
eq adec(pubK(A), aenc(priK(A),DL)) = DL .
```

which states that a ciphertext encrypted by the private key of A can be decrypted by the public key of A. We turn to model exchanged messages. The two messages are represented by the following operators:

```
op msg1 : Prin Prin Prin DataL Nat -> Msg {constr}
op msg2 : Prin Prin Prin DataL Nat -> Msg {constr}
```

The first, second, and third arguments are the real author, the seeming sender, and the recipient of a given message, respectively. `Nat` is the sort of natural numbers. Given three principals a, b, a', a ciphertext c, and a number t, `msg1`(a', a, b, c, t) denotes a message (i). With this message, b is the recipient of the message, a is the seeming sender whom b believes that he/she is the principal who sent

the message, and a' is the real creator of that message. In particular, when a' is the intruder, the intruder tries to impersonate a to send the message to b. Note that the first argument is used for modeling and verification purposes only, but it can neither be seen by the receiver nor be controlled by the intruder. The argument of the sort `Nat` is embedded in the last of each operator to store the time when the corresponding message is sent. That time information is represented as a natural number. Initially, the time of the system is set to 0, and after each action such as a message being sent, it is incremented.

We turn to model the protocol execution. Sorts `Sys` and `Network` are defined, representing the state space and the network, where the network is modeled as an associative-commutative collection (AC-collection) of messages exchanged between principals. All initial states are represented by the constant `init`. Four observers `nw`, `usecret`, `knl`, and `time` are defined, observing the network, the set of secret keys used by all principals, the knowledge of the intruder, and the time of the system, respectively. Their declarations and the definition of initial states are as follows:

```
op init    :      -> Sys {constr}
op nw      : Sys -> Network
op usecret : Sys -> SecretSet
op knl     : Sys -> DataL
op time    : Sys -> Nat
eq nw(init)      = void .
eq usecret(init) = empty .
eq time(init)    = 0 .
eq knl(init)     = (priK(intru) || pubK(intru)) .
```

where `SecretSet` is the sort of sets of secret keys. The four equations say that in an initial state, the network is empty (denoted by `void`), the set of secret keys used is also empty, the time of the system is 0, and the intruder knowledge is their own private and public keys.

We define two transitions modeling how messages (i) and (ii) are sent by principals. The transition `send1` below models how A sends a message (i) to B:

```
op send1 : Sys Prin Prin Secret -> Sys {constr}
ceq nw(send1(S,A,B,K))  = msg1(A,A,B,
    aenc(pubK(B), aenc(priK(A), B || K)), time(S))
    , nw(S)
 if c-send1(S,K) .
ceq usecret(send1(S,A,B,K)) = (K usecret(S))
 if c-send1(S,K) .
ceq time(send1(S,A,B,K))    = s(time(S))
 if c-send1(S,K) .
ceq knl(send1(S,A,B,K)) =
  (aenc(pubK(B), aenc(priK(A), B || K)) || knl(S))
 if c-send1(S,K) .
-- the state remains unchanged
ceq send1(S,A,B,K) = S if not c-send1(S,K) .
-- effective condition
eq c-send1(S,K) = not(K \in usecret(S)) .
```

where `A`, `B`, `K`, and `S` are CafeOBJ variables of the corresponding sorts. `\in` is the membership predicate. `c-send1` is the effective condition of the transition. The first four conditional equations say that if the secret K has not been used, `A` uses it to construct a message (i), sends the message to `B`, the secret is put into the set of secrets used, the time of the system is incremented, and the ciphertext sent to B is appended to the intruder knowledge (i.e., the intruder learned the ciphertext). In a similar way, we define another transition modeling how B sends a message (ii) to A.

We model the intruder with the Dolev-Yao's capabilities [15]. Precisely, the intruder can:

(1) intercept any message sent in the network and learn the information carried in that message;
(2) randomly choose a secret value and use it for something else;
(3) use any cryptographic primitive function with any available information as input to learn the output; and
(4) use the available information to construct a message, and impersonate an honest principal to send the message to another.

The previously described transition `send1` illustrated how the Dolev-Yao capability (1) is fulfilled, i.e., whenever an honest principal sends a message (i) to another, the intruder learns the ciphertext carried in that message. We show part of another transition, illustrating how the Dolev-Yao capability (3) is fulfilled:

```
op g2 : Sys Data DataL -> Sys {constr}
ceq knl(g2(S,D1,DL2)) = (aenc(D1,DL2) || senc(D1,DL2) ||
    knl(S)) if c-g2(S,D1,DL2) .
eq c-g2(S,D1,DL2) = (D1 \in knl(S) and DL2 \in knl(S)) .
```

It states that if two pieces of information `D1` and `DL` are available to the intruder, the intruder can use `D1` as a key to symmetric encrypt and asymmetric encrypt `DL` and learn the two obtained ciphertexts.

2) FORMAL VERIFICATION

We verify the *key secrecy* property, which states that a secret key can be securely distributed to principals. We specifies the property as the following state predicate:

```
op keySe : Sys Prin Prin Secret Nat -> Bool
eq keySe(S,A,B,K,N) = (not(A = intru or B = intru) and
    msg1(A,A,B, aenc(pubK(B), aenc(priK(A), B || K)), N)
     \in nw(S))
  implies not(K \in knl(S)) .
```

It states that if honest principal `A` has sent to honest principal `B` a secret key `K` through a message (i), that is, the message exists in the network, then the intruder is unable to learn `K`, that is, the secret key does not exist in the intruder knowledge. We prove that `keySe` is invariant based on the simultaneous induction method, by using the tool IPSG to produce the so-called *proof score*. IPSG was implemented in Maude [32] and uses CafeInMaude [42], which is the second major implementation of CafeOBJ in the Maude environment. We refer the readers to [10] for the tool implementation, how to use it, and how it works. We first ask IPSG to generate the proof score attempt of `keySe`. In that generated proof, there exist some sub-cases in which we need to conjecture lemmas to discharge the sub-cases. We then use IPSG to produce the proof of `keySe` again as well as the proof attempt of

such lemmas, which may require us to conjecture some other lemmas. The process is repeated until no new lemma is required. In the following, we describe part of the proof score of keySe, when a new lemma is needed, and how we can construct such a lemma.

Recall that keySe is proved by induction. The proof of the base case is done through a so-called CafeOBJ open-close fragment as follows:

```
open INV .
  ops a b : -> Prin .  op k : -> Secret .  op n : -> Nat .
  red keySe(init,a,b,k,n) .
close
```

INV is the CafeOBJ module containing the protocol specification and the predicate keySe. a and b, which are called fresh constants, denote two arbitrary principals; k and n can be understood likewise. Feeding this open-close fragment into CafeOBJ, true is returned, and then the case is discharged. The proof score of keySe is a collection of open-close fragments like that one.

In the proof of the induction case in which send1 is taken into account, we try to prove the following:

```
keySe(s,a,b,k,n) implies keySe(send1(s,r1,r2,r3),a,b,k,n)
```

where r1 and r2 are fresh constants of Prin, denoting arbitrary principals; s and r3 are fresh constants of Sys and Secret, respectively. However, CafeOBJ returns a complex term for that implication. In such a case, case splitting is typically used to split the case into some sub-cases, where each sub-case is discharged by a corresponding open-close fragment. Let us consider another open-close fragment in the proof attempt of keySe produced by IPSG:

```
open INV .
  ops a b r1 r2 : -> Prin .   ops k r3 : -> Secret .
  op n : -> Nat .             op s : -> Sys .
  eq (r3 \in usecret(s)) = false .
  eq a = r1 .   eq b = r2 .   eq k = r3 .
  eq (r1 = intru) = false .   eq (r2 = intru) = false .
  eq time(s) = n .            eq (r3 \in knl(s)) = true .
  eq (msg1(r1,r1,r2,aenc(pubK(r2),aenc(priK(r1),(r2 ||
     r3))),n) \in nw(s)) = false .
  red keySe(s,a,b,k,n) implies
     keySe(send1(s,r1,r2,r3),a,b,k,n) .
close
```

The equations characterize the sub-case. For instance, the Boolean term r3 \in usecret(s) is used to split the induction case into two sub-cases: (1) it is true and (2) it is false. The first equation considers sub-case (2). However, CafeOBJ returns false for the open-close fragment. Provided that keySe is invariant as our expectation, what can be deduced is that states denoted by the fresh constant s must be unreachable. There should exist a contradiction among the equations characterizing the sub-case. From our comprehension of the protocol, we strongly believe that if a secret key exists in the intruder knowledge, it should exist in the set of used secrets. Consequently, it turns out that the following two equations are contradicted:

```
  eq (r3 \in usecret(s)) = false .
  eq (r3 \in knl(s)) = true .
```

Based on that deduction, a lemma, namely inv2, is conjectured as follows:

```
op inv2 : Sys Secret -> Bool
eq inv2(S,K) = K \in knl(S) implies K \in usecret(S) .
```

It says that every secret K available to the intruder exists in the set of used secrets. Then, in the open-close fragment above, actually, the following reduce command is used instead:

```
red inv2(s,r3) implies keySe(s,a,b,k,n) implies
    keySe(send1(s,r1,r2,r3),a,b,k,n) .
```

true is now returned for the open-close fragment. We say that inv2 is used as a lemma.

The remaining part of the keySe's proof score (i.e., a collection of open-close fragments) uses some other lemmas as well. To complete the formal verification, we also need to prove inv2 and those other lemmas. Note that, if all of those lemmas are available, IPSG will generate the complete proof scores for all of them including keySe. All proof scores of keySe and the lemmas as well as the inputs for IPSG to produce those proof scores again are available on the webpage.[3] Even though the verification process is not completely automated, IPSG indeed helps us to save time and effort, and more importantly to avoid human errors, which could happen if we write proof scores by hand.

IV. POST-QUANTUM SSH TRANSPORT LAYER PROTOCOL

PQ SSH [7] relies on the hybrid key exchange method, i.e., a classical key exchange algorithm and a quantum-resistant Key Encapsulation Mechanism (KEM) are used in parallel. KEMs can be regarded as a new formulation of key exchange algorithms according to the NIST standardization project.[4] The following is the general definition of KEMs.

Definition 3: A key encapsulation mechanism (KEM) 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 secret key sk.
- Encaps(pk) $\rightarrow$ (c, k): A probabilistic *encapsulation* algorithm that takes as input a public key pk, and outputs an encapsulation (or ciphertext) c and a shared secret $k \in \mathcal{K}$.
- Decaps(c, sk) $\rightarrow$ k: A (usually deterministic) *decapsulation* algorithm that takes as inputs a ciphertext c and a secret key sk, and outputs a shared secret $k \in \mathcal{K}$.

There are different approaches to post-quantum public-key cryptographic algorithm construction, such as lattice-based, hash-based, and code-based. The latticed-based approach tends to be the most promising way to construct future post-quantum KEMs as the number of latticed-based submissions is the most in the NIST standardization competition. In this approach, a post-quantum algorithm can base its security on

[3] https://github.com/duongtd23/PQSSH
[4] https://csrc.nist.gov/projects/post-quantum-cryptography

the difficulty of learning with errors, ring learning with errors, and learning with rounding, among others. For example, a KEM based on learning with errors generally computes the public key $pk = A * s + e$, where A is a public matrix, s is a vector of coefficients serving as a secret key, and e is a small error vector of coefficients acting as a noise. This public key is sent to Bob with the expectation that Eve is unable to derive $A * s$ (and subsequently, s) even though A and pk are given. $A * s$ can be interpreted as a vector in the lattice $\mathcal{L}\{a_1, \ldots, a_k\}$. Because e is small, pk is close to $A*s$. Thus, recovering $A*s$ corresponds to finding the closest vector problem in lattices. No efficient algorithm on either classical computers or quantum computers is known to solve the closest vector problem when the lattice dimension is large, so it is believed hard even for quantum computers. Various latticed-based KEMs can be named, such as CRYSTALS-Kyber [8], [43], Saber [44], FrodoKEM [45], NTRU [46], and NTRU Prime [47]. In the IETF Draft specifying PQ SSH [7], the quantum-resistant KEM is fixed to CRYSTALS-Kyber, while the classical key exchange algorithm used is ECDH.

The messages exchanged in the PQ SSH protocol are depicted in Fig. 2. Each server host B owns a public host key (LK_B) and a private host key (LSK_B), where the public host key is known by all clients. To initialize a new connection between client A and server B, a pair of VERSION_EX messages is sent by them, exchanging the protocol versions on each side. The message can be called the version exchange message. Then, they exchange a pair of KEX_ALGR messages, indicating their supported algorithms (cryptographic primitives) sorted in order of preference. The message can be called the key exchange algorithms message. After that A generates: (1) an ECDH ephemeral key pair, i.e., secret key and its associated public key ($\text{ECDH}_{\text{PK}_A}$), and (2) a KEM public key (KEM_{PK_A}), i.e., the output of the algorithm `KeyGen`. A then sends the two public keys to B through a KEX_HBR_INIT message (key exchange initiation message). Upon receiving that KEX_HBR_INIT message, B also generates an ECDH ephemeral key pair and performs the algorithm `Encaps` to get a ciphertext (KEM_{C_B}). B then replies back to A with a KEX_HBR_REPLY message (key exchange reply message), consisting of the public host key of B, the ECDH ephemeral public key, the ciphertext, and a signature of the ''exchange hash.'' The precise computations of the exchange hash and the signature are depicted in Fig. 3, where hash denotes the hash function. First, the shared secret K is computed by hashing the concatenation of the ECDH and KEM shared secrets. Then, the exchange hash H is computed by hashing the concatenation of the payloads of the two VERSION_EX messages and the two KEX_ALGR messages, the public host key of B, the ECDH and KEM ephemeral public keys of the client, the ECDH ephemeral public key and the KEM ciphertext of the server, and the shared secret K. Afterward, the signature SIGN is computed by signing H under B's private host key.

V. MODELING THE PROTOCOL

This section presents how to model the protocol in CafeOBJ. We also explain the threat model used and how to specify it in the CafeOBJ formal specification.

A. MODELING ECDH AND KEMs

We introduce CafeOBJ sorts `EcSecretK`, `EcPublicK`, and `EcShareK` representing ECDH secret keys, public keys, and shared secrets, respectively, and some operators as follows:

```
[EcSecretK EcPublicK EcShareK]
-- associated ECDH public key is derived from secret key
op ecPublic : EcSecretK           -> EcPublicK {constr}
-- a shared key is computed from a public & a secret keys
op ecShare  : EcPublicK EcSecretK -> EcShareK
-- constructor of a shared key is a secret key pair
op _|_  : EcSecretK EcSecretK -> EcShareK  {constr comm}
```

The first operator represents the computation of the associated public key from a secret key. The second operator represents the computation of the shared secret from a public key and a secret key. Let `PK` and `K` be CafeOBJ variables of the sorts `EcPublicK` and `EcSecretK`, respectively, the semantic of `ecShare` is defined by the following equation:

```
eq ecShare(PK,K) = (ecSecret(PK) | K) .
```

where `ecSecret` returns the associated secret key of a given public key (it is the projection function of `ecPublic`). Given two ECDH secret keys k_1 and k_2, $\texttt{ecPublic}(k_1)$ denotes the public key associated with k_1, and $(k_1 \,|\, k_2)$ denotes the shared secret obtained from that public key and the secret key k_2. The operator $\_\,|\,\_$ is commutative, namely $(k_1 \,|\, k_2)$ and $(k_2 \,|\, k_1)$ are identical, thanks to the CafeOBJ attribute `comm`.

We choose to model KEMs based on their general definition, i.e., Definition 3. The CafeOBJ formal specification of the protocol does not take into account how CRYSTALS-Kyber KEM is implemented, that is, we omit to specify its implementation components in detail, such as vectors and matrices. The three algorithms `KeyGen`, `Encaps`, and `Decaps` are regarded as three black boxes taking some inputs and returning the outputs. Using abstract versions of cryptographic primitives to model them like this is commonly made in the symbolic analysis of cryptographic protocols. For example, to model hash functions, typically, it suffices to use just a function/operator, which takes any data as input and returns the corresponding hash. The design and implementation aspects of the hash functions, such as block cipher, would be omitted in the formal specification.

To model KEMs in CafeOBJ, sorts `PqSecretK`, `PqPublicK`, `PqShareK`, and `PqCipher` are defined, representing secret keys, public keys, shared secrets, and ciphertexts (or encapsulations), respectively. Let `K'`, `K2'`, and `K3'` are variables of `PqSecretK`, `PK'` and `C` are variables of `PqPublicK` and `PqCipher`, respectively. The algorithms `KeyGen`, `Encaps`, and `Decaps` are modeled by the following operators and equations:

version exchange	VERSION_EX	$A \rightarrow B:$	Version_A
	VERSION_EX	$B \rightarrow A:$	Version_B
key exchange algorithms	KEX_ALGR	$A \rightarrow B:$	Suites_A
	KEX_ALGR	$B \rightarrow A:$	Suites_B
key exchange initiation	KEX_HBR_INIT	$A \rightarrow B:$	$\text{ECDH}_{\text{PK}_A}, \text{KEM}_{\text{PK}_A}$
key exchange reply	KEX_HBR_REPLY	$B \rightarrow A:$	$\text{LK}_B, \text{ECDH}_{\text{PK}_B}, \text{KEM}_{C_B}, \text{SIGN}$

FIGURE 2. Messages exchanged in the PQ SSH protocol.

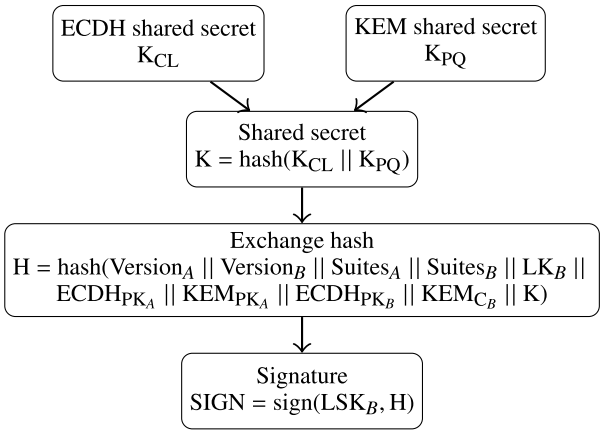

FIGURE 3. Exchange hash and signature calculation.

```
op keygen  : PqSecretK            -> PqPublicK {constr}
-- Encaps: returns ciphertext
op encapsC : PqPublicK PqSecretK -> PqCipher  {constr}
-- Encaps: returns shared key
op encapsK : PqPublicK PqSecretK -> PqShareK
op decaps  : PqCipher  PqSecretK -> PqShareK
-- constructor of a shared key is a secret key pair
op _&_     : PqSecretK PqSecretK -> PqShareK  {constr}
eq encapsK(PK', K') = (pqSecret(PK') & K') .
ceq decaps(C, K')   = (K' & pqSecret(C))
  if (pqPublic(C) = keygen(K')) .
ceq (decaps(C, K') = (K' & K2')) = false
  if not(K2' = pqSecret(C)) .
ceq (decaps(C, K') = (K2' & K3')) = false
  if not(K' = K2') .
```

Note that KeyGen and Encaps are probabilistic algorithms. Thus, to specify them as deterministic procedures in CafeOBJ, an argument of the sort PqSecretK is added as the input argument. Note also that with Encaps, two separate operators encapsC and encapsK are defined, respectively returning the ciphertext and the shared secret. pqPublic and pqSecret are the projection functions of encapsC, returning its first and second arguments, respectively (pqSecret is also the projection function of keygen). Given an encapsulation C and a secret key K', the second equation states that Decaps(C,K') properly outputs the shared secret only if C encapsulates some secret to the associated public key of K' (in other words, the public key of C is the associated public key of K'). The third equation states that Decaps(C,K) cannot be (K' & K2') if C does not encapsulate K2'.

B. MODELING CRYPTOGRAPHIC PRIMITIVES AND MESSAGES EXCHANGED

We introduced sorts Prin, PubKey, PriKey, Version, and Suites representing principals, public host keys, private host keys, protocol versions, and lists of supported algorithms, respectively. Similar to the specification of the SDS protocol in Section III-C, sorts Data and DataL are introduced as the supersorts of all sorts mentioned before, such as EcPublicK and PqPublicK, and || is defined as the concatenation operator. To model the hash function, the Sign and Verify signature algorithms, we declare the following operators:

```
-- hash function
op h      : DataL                          -> Data {constr}
--          key    plaintext                  signature
op sign   : DataL  DataL                    -> Data {constr}
--          key    plaintext  signature
op verify : DataL  DataL      DataL         -> Bool
```

There are several equations defining those operators, for example:

```
eq verify(pubK(A), D, SIGN) = (SIGN = sign(priK(A),D)) .
```

which states that given a public host key of principal A, a message D, and a signature, the Verify algorithm outputs true when the signature is obtained by signing D under the private host key of A.

We turn to model messages exchanged in the protocol. The VERSION_EX, KEX_ALGR, KEX_HBR_INIT, and KEX_HBR_REPLY messages depicted in Fig. 2 are respectively represented by the following operators:

```
op verM    : Prin Prin Prin Version    -> Msg {constr}
op kexAlgM : Prin Prin Prin Suites     -> Msg {constr}
op hbrIniM : Prin Prin Prin DataL Nat  -> Msg {constr}
op hbrRepM : Prin Prin Prin DataL Nat  -> Msg {constr}
```

Recall that Nat is the sort of natural numbers and the first, second, and third arguments of each operator denote the actual sender, the seeming sender, and the recipient of a given message, respectively. For example, with a VERSION_EX message in the form of verM(A2,A,B,V), the second argument, A, is what the receiver B can see about the sender. The actual sender is A2, whom B is unable to observe. A2 may not be A, and if that is the case, A2 (a dishonest principal) is trying to claim to be A to communicate with B. We embed the first argument into each message in order to specify and verify the *authentication* property, which will be reported in Section VI-D.

With the two operators representing KEX_HBR_INIT and KEX_HBR_REPLY messages, we additionally embed an argument of the sort `Nat` in the last of each one to store the time when the corresponding message is sent. The time information is necessary to specify and verify the *forward secrecy* property later on, for instance, to check whether the key concerned is established before the compromise of the server's private host key who established that key in a session with another client. With the first two operators, we could also embed an argument of the sort `Nat` in the last of each one, but saving time information of VERSION_EX and KEX_ALGR messages is not strictly necessary to complete the verification.

C. MODELING THE PROTOCOL EXECUTION

Sorts `Sys` and `Network` are defined, representing the state space and the network, where the network is modeled as an AC-collection of messages exchanged between principals. All initial states are represented by the constant `init`. Five observers `nw`, `usecret`, `time`, `leakscr`, and `knl` are defined, observing the network, the set of ECDH & KEM secret keys used by all principals, the system time, the compromised secrets, and the knowledge of the intruder, respectively. `usecret` is used to guarantee the uniqueness of ephemeral secret keys. The compromised secrets can be ephemeral secret keys, private host keys, and shared secrets between two participants, which will be described in the next sections. The declarations of `init` and the five observers and the definition of initial states are as follows:

```
op init     :      -> Sys {constr}
op nw       : Sys -> Network
op usecret  : Sys -> SecretKS
op time     : Sys -> Nat
op leakscr  : Sys -> SecretKS
op knl      : Sys -> DataL
eq nw(init) = void .
eq usecret(init) = empty .
eq time(init) = 0 .
eq leakscr(init) = empty .
eq knl(init) = (priK(intru) || pubK(intru)) .
```

where `SecretKS` is the sort of sets of secret data types (e.g., private host keys, ECDH & KEM secret keys). From the five equations, it follows that in an initial state, the network is empty (denoted by `void`), the set of secrets used is also empty, time of the system is `0`, no secret is revealed, and the intruder knowledge is their own private and public host keys.

For each of the six messages depicted in Fig. 2, we define a transition modeling how that message is sent. For instance, we model sending a KEX_HBR_INIT message through the following transition:

```
op cHbrInit : Sys Prin Prin Prin EcSecretK PqSecretK
    Version Version Suites Suites -> Sys {constr}
ceq nw(cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2)) =
  (hbrIniM(A,A,B, ecPublic(K) || keygen(K'),
    time(S)) , nw(S))
 if c-cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2) .
ceq usecret(cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2)) =
  (K K' usecret(S))
 if c-cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2) .
ceq time(cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2)) =
  s(time(S))
 if c-cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2) .
ceq knl(cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2))  =
  (ecPublic(K) || keygen(K') || knl(S))
 if c-cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2) .
eq c-cHbrInit(S,B2,A,B,K,K',V,V2,CSs,CSs2) =
 (kexAlgM(A,A,B,CSs) \in nw(S) and
  kexAlgM(B2,B,A,CSs2) \in nw(S) and
  not(K \in usecret(S) or K' \in usecret(S))) .
```

where `A`, `B`, `V`, etc., are CafeOBJ variables of the corresponding sorts. `\in` is the membership predicate. `c-cHbrInit` is the effective condition of the transition, which states that the transition cannot proceed unless two key exchange algorithms messages have been exchanged, and the two secret keys (ECDH and KEM) `K` and `K'` have not been used before. The first four conditional equations say that if the effective condition is satisfied, from the two secret keys `K` and `K'`, client `A` sends the two associated public keys to `B` under a key exchange initiation message (by putting that message into the network), the two secret keys are put into the set of secret keys used, the time is incremented, and the two public keys are added to the intruder knowledge (i.e., the intruder learned the two public keys).

D. THREAT MODEL AND MODELING THE INTRUDER

The threat model used in this verification case study is an extended version of the Dolev-Yao intruder model [15]. As a Dolev-Yao intruder, the intruder can completely control the network, concretely:

(1) The intruder can intercept any message sent in the network and glean information carried in that message. This capability has been partially illustrated through the definition of the transition `cHbrInit` presented in Section V-C. That is, whenever an honest principal sends two public keys to another one through a KEX_HBR_INIT message, the intruder will learn the two public keys.
(2) The intruder also knows all publicity information such as protocol versions, names of cryptographic primitives, and public host keys even without gleaning them from the network.
(3) The intruder can select an ephemeral secret key (either ECDH one or KEM one), and generate the corresponding public key or the shared secret provided that the secret key has not been used before (uniqueness).
(4) If a piece of information is available to the intruder, they can use any cryptographic primitive function taking the information as input and learning the output.
(5) The intruder can use the information available to them to build a message and impersonate some honest principal to send the message to another.

In addition to the Dolev-Yao capabilities above, our threat model also considers the following:

(6) The security of ECDH is broken. If two ECDH public keys are given to the intruder, the intruder can derive the corresponding shared secret, which is assumed by utilizing the power of large quantum computers.
(7) Secrets may be compromised and the intruder gleans them. All of ECDH & KEM ephemeral secret keys, private host keys, and shared secrets established between two principals are possibly revealed.

The following is part of a transition illustrating how the Dolev-Yao capability (5) is fulfilled:

```
op fkHbrInit : Sys Prin Prin EcPublicK PqPublicK -> Sys
     {constr}
ceq nw(fkHbrInit(S,A,B,PK,PK')) =
  (hbrIniM(intru,A,B, PK || PK', time(S)) , nw(S))
 if c-fkHbrInit(S,A,B,PK,PK') .
ceq time(fkHbrInit(S,A,B,PK,PK')) = s(time(S))
 if c-fkHbrInit(S,A,B,PK,PK') .
eq c-fkHbrInit(S,A,B,PK,PK')       =
  (PK \in knl(S) and PK' \in knl(S)) .
```

It specifies how the intruder can forge a key exchange initiation message. The equations straightforwardly state that if the two ECDH and KEM public keys `PK` and `PK'` are in the intruder knowledge, the intruder can impersonate principal `A`, sending the two public keys through a key exchange initiation message to principal `B`. As mentioned before, the first parameter inside `hbrIniM` is `intru` but not `A`, which is impossible to be seen by the receiver.

With the capability (7), we show below part of the transition `lPqSecretK1`, which partially models the compromise of a KEM ephemeral secret key:

```
op lPqSecretK1 : Sys Prin Prin EcSecretK PqSecretK Nat ->
     Sys {constr}
ceq time(lPqSecretK1(S,A,B,K,K',N))     = s(time(S))
 if c-lPqSecretK1(S,A,B,K,K',N) .
ceq leakscr(lPqSecretK1(S,A,B,K,K',N)) = (K' leakscr(S))
 if c-lPqSecretK1(S,A,B,K,K',N) .
ceq knl(lPqSecretK1(S,A,B,K,K',N))      = (K' || knl(S))
 if c-lPqSecretK1(S,A,B,K,K',N) .
eq c-lPqSecretK1(S,A,B,K,K',N) =
  hbrIniM(A,A,B, ecPublic(K) || keygen(K'), N) \in nw(S) .
```

It states that if a key exchange initiation message is in the network, the KEM secret key associated with the public key sent in that message can be compromised. If that is the case, the secret key is added to the intruder knowledge as well as the set of compromised secrets, and the time of the system is incremented. There is another transition modeling the compromise of a KEM secret key through a key exchange reply message.

In summary, the complete formal specification of the PQ SSH protocol consists of 974 lines of CafeOBJ code, where 727 lines are dedicated to specifying the protocol execution and 247 lines are dedicated to defining invariants (and lemmas), which are used for the formal verification. Note that the formal specification limits neither the number of honest principals participating in the protocol nor the number of sessions that the protocol can execute. Generally speaking, it allows any principal (a variable of sort `Prin`) to initialize a session with any other principal regardless of how many times and without any restriction by executing the transition formalizing the sending of a VERSION_EX message.

VI. FORMAL ANALYSIS

Security considerations of the SSH Transport Layer protocol are claimed in [4] as follows:

> This protocol provides a secure encrypted channel over an insecure network. It performs server host authentication, key exchange, encryption, and integrity protection. It also derives a unique session ID that may be used by higher-level protocols.

Such security requirements should be kept fulfilled in the PQ SSH protocol. This section presents the formal analysis of four properties including (1) *session key secrecy*, (2) *forward secrecy*, (3) *session identifier uniqueness*, and (4) *authentication*. (1) makes sure that nobody can learn a shared secret negotiated between a client and a server except those two principals (see Section VI-A). (2) guarantees that even if a private host key of a server is compromised, shared secrets established before the compromise remain secure (see Section VI-B). (3) ensures that the exchange hash, acting as the session identifier, is unique (see Section VI-C). (4) states that upon completion of a protocol execution, if client *A* has communicated apparently with server *B*, then the server is indeed *B* (see Section VI-D). We successfully complete the verifications of (1), (2), and (3), in which 17 additional lemmas are introduced. Whereas, we find a counterexample of (4). In the following, we report in detail the analysis results.

Checking the Specification: We first check that the formal specification allows two principals successfully complete a protocol execution and obtain the shared secret. This must be fulfilled, otherwise, anything we do after is entirely meaningless. We have confirmed that by showing a reachable state satisfying that requirement through a sequence of transitions, which can be found on the webpage mentioned before.

Checking Intruder Capability of Learning ECDH Shared Secret: We also confirmed that the intruder is able to learn the ECDH shared secret established between two honest principals as what has been modeled for the intruder's capabilities. Similarly, we have verified that by pointing out a reachable state in which the intruder can learn such a secret.

A. SESSION KEY SECRECY PROPERTY

The negotiation of a shared secret between two principals must be secure against any third party. This is called the *session key secrecy* property, which is specified by the following predicate:

```
op keySe : Sys Prin Prin Prin Version Version Suites Suites
     EcSecretK PqSecretK EcPublicK PqCipher Data Nat Nat ->
      Bool
eq keySe(S,B2,A,B,V,V2,CSs,CSs2,K,K',PK2,C,SIGN,N,N2) =
 (not(A = intru or B = intru) and
  hbrIniM(A,A,B, ecPublic(K) || keygen(K'), N) \in nw(S)
  and
  hbrRepM(B2,B,A, pubK(B) || PK2 || C || SIGN, N2)
    \in nw(S) and
  verify(pubK(B), h(V || V2 || CSs || CSs2 ||
```

```
    pubK(B) || ecPublic(K) || keygen(K') || PK2 || C ||
    h(ecShare(PK2,K) || decaps(C,K'))),
   SIGN) and
  not(decaps(C,K') \in leakscr(S)) and
  not(K' \in leakscr(S) or pqSecret(C) \in leakscr(S)) and
  not(priK(B) \in' leakscr(S)))
 implies not(h(ecShare(PK2,K) || decaps(C,K')) \in knl(S)).
```

The predicate states that when honest client `A` has sent to honest server `B` a KEX_HBR_INIT message and has received back a KEX_HBR_REPLY message apparently sent from `B` with a valid signature of the exchange hash, neither the KEM shared secret, the two corresponding ephemeral secret keys, nor the private host key of the server is revealed, then the intruder cannot learn the shared secret (i.e., the hash of the ECDH shared secret and the KEM shared secret). From the client `A`'s observation, the KEX_HBR_REPLY message is sent from `B`, that is the only thing `A` can see. `A` cannot determine whether or not there exists another principal `B2`, the actual sender, trying to claim to be `B` to send to him/her the message. That is true in reality: when Alice receives a message seemingly sent from Bob, nothing guarantees that Bob is the actual sender, the message is possibly originated by Eve. `keySe` is formally verified with some additional lemmas and the employment of IPSG. The complete proof is available on the webpage.[5]

All sub-constraints in the premise of the predicate are necessary, namely, if any of them is eliminated, the predicate will be no longer valid. Indeed, if the following constraint:

(1) the signature of the exchange hash in the KEX_HBR_REPLY message is valid

is eliminated, the reply message received may be actually forged by the intruder (`B2` is `intru`), who is trying to impersonate `B`. In this case, the signature will be failingly verified if `A` does a check, however, it is not performed actually because (1) is removed. If the following constraint:

(2) the KEM shared secret is not revealed

is removed, the intruder can derive the shared secret because the intruder can break ECDH's security to learn the ECDH shared secret. It also explains why we only bind the non-reveal of the KEM shared secret in the premise of `keySe`. If the following constraint:

(3) the two corresponding KEM ephemeral secret keys are not revealed

is removed, the intruder can easily derive the KEM shared secret, and then they can derive the shared secret as explained above. If the following constraint:

(4) the private host key of the server is not revealed

is eliminated, the intruder can use the revealed key to sign the exchange hash to make a valid signature. Subsequently, the intruder can completely impersonate server `B` to do the key exchange with `A`. As a result, the shared secret is obviously available to the intruder. Readers can find on the above-mentioned webpage the counterexamples showing that the predicate will be no longer valid if any of (1), (2), (3), and (4) is eliminated.

[5]https://github.com/duongtd23/PQSSH

B. FORWARD SECRECY PROPERTY

Forward secrecy property in general is defined as that the compromise of a long-term private key does not break the secrecy of a session key if the session is completed before the compromise. This property is specified almost similar to the invariant `keySe`. The only difference is that in the premise, instead of **not**`(priK(B) \in' leakscr(S))`, the following constraint is used:

```
priK(B) \in' leakscr(S) and
N2 < timeLeak(priK(B), leakscr(S))
```

It means that the premise allows the private host key of the server is compromised but the compromise must happen after the KEX_HBR_REPLY message is sent. Precisely, the predicate formalizing the *forward secrecy* property states that if honest client `A` has sent to honest server `B` a KEX_HBR_INIT message and has received back a KEX_HBR_REPLY message apparently sent from `B` with a valid signature of the exchange hash, neither the KEM shared secret nor the two corresponding ephemeral secret keys are revealed, the private host key of the server is compromised but the compromise happens after the KEX_HBR_REPLY message is sent, then the intruder cannot learn the shared secret. It guarantees that even if the server's private host key is compromised, shared secrets negotiated before the compromise remain secure. The property is also formally verified with the employment of IPSG.

Similar to `keySe`, we cannot eliminate any constraint in the premise of the predicate because of the same reasons explained previously. The three above-mentioned constraints (1), (2), and (3) are compulsory assumptions to guarantee the secrecy of a shared secret, that is, the signature of the exchange hash in the reply message received must be correctly verified and the KEM shared secret & the two KEM ephemeral secret keys must be not compromised. While with the server's private host key, we need to require that either the key is uncompromised or the compromise happens after the sending of the reply message.

C. SESSION IDENTIFIER UNIQUENESS PROPERTY

During the key negotiation between two principals, the server authenticates himself/herself by signing the exchange hash with his/her private host key and sending the signature to the client. Besides that purpose, the exchange hash is also used as the session identifier for this connection. This session identifier must be unique in order to be used by some higher-level protocols as claimed in [4]. We also formally verified this property.

D. AUTHENTICATION PROPERTY

The IETF Draft [7] states that the protocol provides server authentication. This property is stated (from a client's point of view) as follows: if client A performs a key negotiation apparently with server B, then the server that A communicates with is really B. We attempt to specify this property in CafeOBJ by the following predicate:

```
op auth : Sys Prin Prin Prin Version Version Suites Suites
  EcSecretK PqSecretK EcPublicK PqCipher Data Nat Nat Nat
    -> Bool
eq auth(S,B2,A,B,V,V2,CSs,CSs2,K,K',PK2,C,SIGN,N,N2,?M) =
 (not(A = intru or B = intru) and
  not(decaps(C,K') \in leakscr(S)) and
  not(K' \in leakscr(S) or pqSecret(C) \in leakscr(S)) and
  not(priK(B) \in' leakscr(S)) and
  hbrIniM(A,A,B, ecPublic(K) || keygen(K'), N) \in nw(S)
  and
  hbrRepM(B2,B,A, pubK(B) || PK2 || C || SIGN, N2)
    \in nw(S) and
  verify(pubK(B), h(V || V2 || CSs || CSs2 ||
    pubK(B) || ecPublic(K) || keygen(K') || PK2 || C ||
    h(ecShare(PK2,K) || decaps(C,K'))),
   SIGN))
 implies
  hbrRepM(B,B,A, pubK(B) || PK2 || C || SIGN, ?M)
    \in nw(S) .
```

where `?M` indicates that this variable is existentially quantified. This `?`-symbol prefix is not mandatory from a syntactic point of view, but it helps to distinguish variables that are existentially quantified and universally quantified. The predicate states that if honest client `A` has sent to honest server `B` a KEX_HBR_INIT message and has received back a KEX_HBR_REPLY message apparently sent from `B` with a valid signature of the exchange hash, neither the KEM shared secret, the two corresponding ephemeral secret keys, nor the private host key of the server is revealed, then `B` has indeed sent the KEX_HBR_REPLY message to `A` at some time denoted by `?M`. Recall that `hbrRepM(B2,B,A,...)` states that from `A`'s point of view, `B` is the sender of the KEX_HBR_REPLY message received; the actual sender is `B2`, which may or may not be `B` and cannot be observed by `A`. Note that it is incorrect to affirm `B2 = B` in the conclusion of `auth`, which is because the capabilities of the intruder allow them to replay any message in the network. It is possible that there exist two different messages in the network, an original one is sent by `B` and the other one is made by the intruder by replaying that original message (in this case, `B2` is the intruder). `?M` must be used because the time when the original message was sent by the honest server `B` is unknown, in particular, it cannot be derived from the time when the replaying message was sent by the intruder (i.e., `N2`).

Step-1	A	$A \rightarrow B$:	$\text{ECDH}_{\text{PK}} \,\|\|\, \text{KEM}_{\text{PK}}$
Step-2	I	learns	$\text{ECDH}_{\text{PK}} \,\|\|\, \text{KEM}_{\text{PK}}$
Step-3	I	$A_2 \rightarrow B$:	$\text{ECDH}_{\text{PK}} \,\|\|\, \text{KEM}_{\text{PK}}$
Step-4	B	$B \rightarrow A_2$:	$\text{LK}_B \,\|\|\, \text{ECDH}_{\text{PK}_2} \,\|\|\, \text{KEM}_{\text{C}} \,\|\|\, \text{SIGN}$
Step-5	I	learns	$\text{LK}_B \,\|\|\, \text{ECDH}_{\text{PK}_2} \,\|\|\, \text{KEM}_{\text{C}} \,\|\|\, \text{SIGN}$
Step-6	I	$B \rightarrow A$:	$\text{LK}_B \,\|\|\, \text{ECDH}_{\text{PK}_2} \,\|\|\, \text{KEM}_{\text{C}} \,\|\|\, \text{SIGN}$

where I denotes the intruder

FIGURE 4. **Counterexample of** `auth`**.**

However, a counterexample of `auth` is found, namely, the property stated by `auth` does not hold. The counterexample can be found on the webpage mentioned before. Fig. 4 briefly explains why the counterexample can happen. According to the figure, there are mainly six steps, where the first name at each step denotes the principal who performs the given action. In the first and second steps, A sends two ephemeral public keys to B under a KEX_HBR_INIT message, and then the intruder gleans them. Using the two public keys gleaned, the intruder tries to impersonate another client A_2 to send them to B (Step-3). In the next two steps, B replies back to A_2 a KEX_HBR_REPLY message with the public host key, an ECDH ephemeral public key, a KEM ciphertext, and a signature over the exchange hash, and then the intruder gleans all of those pieces of information. In the final step, by using the information just learned, the intruder tries to impersonate B to send a KEX_HBR_REPLY message to A. After this step, there exists a valid KEX_HBR_REPLY message whose creator is the intruder, the seeming sender is B, and the receiver is A in the network (that message is `hbrRepM(intru,B,A,...)`), but there does not exist a KEX_HBR_REPLY message with the same content really sent by B to A in the network. B sent such a message to A_2 instead.

Revising the Exchange Hash: The found counterexample can be regarded as a weakness of the protocol. To address the weakness and to make the protocol enjoy the *authentication* property, we proposed to revise the protocol by including the identifiers of the client and the server in the exchange hash. Precisely, the exchange hash is computed as follows:

$$\text{H} = \text{hash}(\text{Version}_A \,||\, \text{Version}_B \,||\, \text{Suites}_A \,||\, \text{Suites}_B \,||\, \text{LK}_B \,||\, \text{ECDH}_{\text{PK}_A} \,||\, \text{KEM}_{\text{PK}_A} \,||\, \text{ECDH}_{\text{PK}_B} \,||\, \text{KEM}_{\text{C}_B} \,||\, \text{K} \,||\, A \,||\, B)$$

The exchanged messages in the improved version of the protocol are kept as depicted in Fig. 2. The computations of the secrets except for the exchange hash are kept as depicted in Fig. 3. With this improved protocol, when the intruder tries to impersonate B to send the KEX_HBR_REPLY message to A (Step-6 in Fig. 4), A will not accept that message because the signature SIGN will not be successfully verified. Upon reception of that message, A expects that the identifiers of A are included in the signature, but actually, SIGN is signed over A_2 rather than A. Therefore, the counterexample will be prevented.

That is the informal argument, to prove that revising the protocol in that way does indeed make it enjoy the *authentication* property, formal verification must be conducted. We revise the CafeOBJ formal specification accordingly and verify the *authentication* property again. With the improved version, we successfully prove `auth` with the employment of IPSG. Besides, the three other properties remain secure with respect to the improved protocol. Note that we need to slightly revise the four predicates specifying the four properties to make the client and the server identifiers included in the exchange hash. Again, the proof scores with respect to the improved protocol are available on the webpage.[6]

The 17 auxiliary lemmas can be reused to formally verify again the *session key secrecy*, *forward secrecy*, and *session*

[6]https://github.com/duongtd23/PQSSH

identifier uniqueness properties with respect to the improved protocol. To this end, we need to slightly revise 5 among those lemmas by simply adding the identifiers of the client and the server concerned into the exchange hash. No new lemma is needed. The remaining job is just to ask IPSG to produce the proofs again for the three properties and the lemmas. The experimental results, which are available on the above-mentioned webpage, indicate that IPSG took around 1 second up to around 8 seconds to produce each invariant proof score, which is reasonably small. Therefore, the verification process helps us to save a lot of time and effort. Once a property has been successfully proved, when the protocol and/or the property are slightly changed, the verification by proof scores allows us to reuse most of the auxiliary lemmas, while the others are only needed to be slightly revised. Regenerating the proofs is trivial because it is automated by the tool taking only a bit of time. This is an advantage of the verification approach compared to model checking-based and its variant approaches. When conducting model checking, each time the protocol or the properties under verification are changed even a little bit, verification should be redone from the beginning, meaning that it is time-consuming because the model checker takes time to terminate for each verification experiment.

Assessment of the Weakness: The counterexample we have found for the *authentication* property does not affect the confidentiality of session keys shared by honest participants, and so it is unreasonable to regard the counterexample as an attack. Indeed, I cannot learn the shared secret derived from the keys generated by A and B depicted in Fig. 4. The intruder could only learn the public information sent in the messages intercepted and forward/replay them to someone, but could not derive the associated secret information, such as the KEM secret key and shared key.

Although the weakness has nothing to do with the leakage of any confidential pieces of information, it would be preferable to fix it. Every cryptographic protocol should be designed such that a protocol execution is completed if and only if designed actions for each entity involved occurred and designed information security objectives, such as *authentication*, are fulfilled. Otherwise, there is something wrong with respect to the protocol. Anything that is beneficial to the information security objectives should not be lost even if it does not cause a compromise of confidential information. Otherwise, trust in the protocol may be discarded or such a weakness of the protocol may be utilized to lead to a more sensitive security flaw in the future. Therefore, we have proposed to fix PQ SSH as presented and formally verified that the improved version of PQ SSH enjoys the *authentication* property as well as the other properties by generating proof scores with IPSG and running them with CafeOBJ.

VII. CONCLUSION

We have formally specified the post-quantum hybrid key exchange SSH Transport Layer protocol [7] in CafeOBJ and conducted the formal analysis with four security properties including (1) *session key secrecy*, (2) *forward secrecy*, (3) *session identifier uniqueness*, and (4) *authentication*. The analysis has formally verified that the protocol enjoys (2), (3), and (4), while it does not enjoy (4). The protocol was then proposed to be slightly improved by adding the identifiers of the client & server into the exchange hash. The formal verification has confirmed that the improved protocol enjoys (4). The properties have been proved with respect to an unbounded number of protocol participants and session executions, by using the tool IPSG to produce the proof scores in CafeOBJ. Even though the verification process is not completely automated, the use of IPSG allows us to focus on only one task, namely to conjecture lemmas. The reuse of most auxiliary lemmas has helped us to save a lot of time and effort when conducting verifications again for the three properties (1), (2), and (3) with respect to the improved protocol.

Comprehending the protocol is the task that takes the most time to conduct the formal analysis. The most difficult task would be to comprehend the post-quantum cryptographic primitives used in the protocol, such as CRYSTALS-Kyber KEM. That is a challenge in the verification/analysis of post-quantum cryptographic protocols compared to classical ones. After understanding such primitives, another challenge is to find a way to model them for formal verification, such as by abstracting KEMs as we did. To come up with a reasonable and strong threat model is also a creative task. Attackers must have quantum-based power, such as breaking classical key exchange algorithms, and must have powerful capabilities, such as fully controlling the network. To find out how to specify in CafeOBJ the attacker's capability of fully controlling the network is also a challenging task.

In spite of the non-trivial contributions, there still exists a limitation in our work. Regarding the threat model in the quantum era that we have used in this work, essentially, the novelty of the intruder capabilities is only the assumption of breaking classical key exchange algorithms such as ECDH. The other considerations are seen as non-novel things from the perspective of standard cryptographic protocol analysis. There exist some more proposals by some other IETF working groups to standardize new post-quantum cryptographic protocols, among them including a quantum-resistant version of the Internet Key Exchange Protocol Version 2 (IKEv2) [48] and the post-quantum OpenPGP [49]. As a piece of our future work, we are interested in conducting formal verification/analysis of these protocols, in which we expect that some more non-trivial capabilities for the quantum attacker will be included.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Nov. 1994, pp. 124–134.

[2] D. Ott and C. Peikert, "Identifying research challenges in post quantum cryptography migration and cryptographic agility," Tech. Rep., 2019.

[3] C. M. Lonvick and T. Ylonen, *The Secure Shell (SSH) Protocol Architecture*, document RFC 4251, Jan. 2006.

[4] C. M. Lonvick and T. Ylonen, *The Secure Shell (SSH) Transport Layer Protocol*, document RFC 4253, Jan. 2006.
[5] C. M. Lonvick and T. Ylonen, *The Secure Shell (SSH) Authentication Protocol*, document RFC 4252, Jan. 2006.
[6] C. M. Lonvick and T. Ylonen, *The Secure Shell (SSH) Connection Protocol*, document RFC 4254, Jan. 2006.
[7] P. Kampanakis, D. Stebila, and T. Hansen, *Post-Quantum Hybrid Key Exchange in SSH*, document Internet-Draft Draft-Kampanakis-Curdle-SSH-PQ-KE-01, Internet Engineering Task Force, Work in Progress, Apr. 2023.
[8] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS–kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, London, U.K., Apr. 2018, pp. 353–367.
[9] R. Diaconescu and K. Futatsugi, *CafeOBJ Report—The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification* (AMAST Series in Computing), vol. 6. Singapore: World Scientific, 1998.
[10] D. D. Tran and K. Ogata, "Formal verification of TLS 1.2 by automatically generating proof scores," *Comput. Secur.*, vol. 123, Dec. 2022, Art. no. 102909.
[11] K. Ogata and K. Futatsugi, "Proof scores in the OTS/CafeOBJ method," in *Proc. 6th IFIP WG 6.1 Int. Conf. Formal Methods Open Object-Based Distrib. Syst. (FMOODS)*, in Lecture Notes in Computer Science, vol. 2884, E. Najm, U. Nestmann, and P. Stevens, Eds., Paris, France: Springer, Nov. 2003, pp. 170–184.
[12] K. Ogata and K. Futatsugi, "Compositionally writing proof scores of invariants in the OTS/CafeOBJ method," *J. Universal Comput. Sci.*, vol. 19, pp. 771–804, Jun. 2013.
[13] E. Rescorla and T. Dierks, *The Transport Layer Security (TLS) Protocol Version 1.2*, document RFC 5246, Aug. 2008.
[14] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3*, document RFC 8446, Aug. 2018.
[15] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 2, pp. 198–207, Mar. 1983.
[16] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput. (STOC)*, 1996, pp. 212–219.
[17] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek, and A. Otmani, "Formal specification and model checking of saber lattice-based key encapsulation mechanism in Maude," in *Proc. 34th Int. Conf. Softw. Eng. Knowl. Eng.*, Madrid, Spain, Jul. 2022, pp. 16–31.
[18] B. Blanchet, "Security protocol verification: Symbolic and computational models," in *Proc. POST*, vol. 7215. Cham, Switzerland: Springer, 2012, pp. 3–29.
[19] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and ProVerif," *Found. Trends Privacy Secur.*, vol. 1, nos. 1–2, pp. 1–135, 2016.
[20] C. Jacomme, E. Klein, S. Kremer, and M. Racouchot, "A comprehensive, formal and automated analysis of the EDHOC protocol," in *Proc. 32nd USENIX Secur.*, Aug. 2023, pp. 5881–5898.
[21] A. Hülsing, K.-C. Ning, P. Schwabe, F. Weber, and P. R. Zimmermann, "Post-quantum WireGuard," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 304–321.
[22] G. Selander, J. P. Mattsson, and F. Palombini, *Ephemeral Diffie-Hellman Over COSE (EDHOC)*, document Internet-Draft draft-IETF-lake-EDHOC-17, Internet Engineering Task Force, Oct. 2022.
[23] V. Cheval, C. Jacomme, S. Kremer, and R. Künnemann, "SAPIC+: Protocol verifiers of the world, unite!" in *Proc. 31st USENIX Secur.*, Aug. 2022, pp. 3935–3952.
[24] B. Blanchet, "An efficient cryptographic protocol verifier based on prolog rules," in *Proc. 14th IEEE Comput. Secur. Found. Workshop*, Jun. 2001, pp. 82–96.
[25] B. Blanchet, V. Cheval, and V. Cortier, "ProVerif with lemmas, induction, fast subsumption, and much more," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 69–86.
[26] D. Basin, C. Cremers, J. Dreier, and R. Sasse, "Symbolically analyzing security protocols using tamarin," *ACM SIGLOG News*, vol. 4, no. 4, pp. 19–30, Nov. 2017.
[27] J. A. Donenfeld, "WireGuard: Next generation kernel network tunnel," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–12.
[28] S. Escobar, C. Meadows, and J. Meseguer, *Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties*. Berlin, Germany: Springer, 2009, pp. 1–50.
[29] C. J. F. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Proc. Int. Conf. Comput. Aided Verification*, vol. 5123, A. Gupta and S. Malik, Eds., Princeton, NJ, USA, Cham, Switzerland: Springer, Jul. 2008, pp. 414–418.
[30] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, *ProVerif 2.05: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*, 2023.
[31] S. Escobar, J. Meseguer, and P. Thati, "Narrowing and rewriting logic: From foundations to applications," in *Proc. 15th Workshop Funct. (Constraint) Log. Program. (WFLP)*, vol. 177, F. J. López-Fraguas, Eds. Madrid, Spain. Amsterdam, The Netherlands: Elsevier, Nov. 2006, pp. 5–33.
[32] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, Eds., *All About Maude* (Lecture Notes in Computer Science), vol. 4350. Cham, Switzerland: Springer, 2007.
[33] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman, "Strand spaces: Why is a security protocol correct?" in *Proc. IEEE Symp. Secur. Privacy*, Oakland, CA, USA, May 1998, pp. 160–171.
[34] S. Escobar, C. A. Meadows, and J. Meseguer, "State space reduction in the Maude-NRL protocol analyzer," in *Proc. Eur. Symp. Res. Comput. Secur.*, vol. 5283, Málaga, Spain, Oct. 2008, pp. 548–562.
[35] S. Escobar, C. Meadows, J. Meseguer, and S. Santiago, "State space reduction in the maude-NRL protocol analyzer," *Inf. Comput.*, vol. 238, pp. 157–186, Nov. 2014.
[36] E. Poll and A. Schubert, "Verifying an implementation of SSH," in *Proc. 7th Int. Workshop Issues Theory Secur. (WITS') Co-Located With ETAPS*, Braga, Portugal Mar. 2007, pp. 164–177.
[37] P. Fiterău-Broştean, T. Lenaerts, E. Poll, J. de Ruiter, F. Vaandrager, and P. Verleg, "Model learning and model checking of SSH implementations," in *Proc. 24th ACM SIGSOFT Int. SPIN Symp. Model Checking Softw.*, Santa Barbara, CA, USA, Jul. 2017, pp. 142–151.
[38] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu, "Cryptographically verified implementations for TLS," in *Proc. 15th ACM Conf. Comput. Commun. Secur.*, Oct. 2008, pp. 459–468.
[39] K. Bhargavan, C. Fournet, R. Corin, and E. Zălinescu, "Verified cryptographic implementations for TLS," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 1–32, Mar. 2012.
[40] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Beguelin, K. Bhargavan, J. Pan, and J. K. Zinzindohoue, "Implementing and proving the TLS 1.3 record layer," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 463–482.
[41] D. E. Denning and G. M. Sacco, "Timestamps in key distribution protocols," *Commun. ACM*, vol. 24, no. 8, pp. 533–536, Aug. 1981.
[42] A. Riesco, K. Ogata, and K. Futatsugi, "A Maude environment for CafeOBJ," *Formal Aspects Comput.*, vol. 29, no. 2, pp. 309–334, Mar. 2017.
[43] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: Algorithm specifications and supporting documentation (version 3.02)," Tech. Rep., 2021.
[44] J. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 10831, A. Joux, A. Nitaj, and T. Rachidi, Eds. Marrakesh, Morocco: Springer, May 2018, pp. 282–305, doi: 10.1007/978-3-319-89339-6_16.
[45] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, and A. Raghunathan, "FrodoKEM: Learning with errors key encapsulation," Tech. Rep., 2021.
[46] A. Hülsing, J. Rijneveld, J. M. Schanck, and P. Schwabe, "High-speed key encapsulation from NTRU," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.*, vol. 10529, W. Fischer and N. Homma, Eds., Taiwan. Cham, Switzerland: Springer, Sep. 2017, pp. 232–252.
[47] D. J. Bernstein, B. B. Brumley, M.-S. Chen, C. Chuengsatiansup, T. Lange, A. Marotzke, B.-Y. Peng, N. Tuveri, C. van Vredendaal, and B.-Y. Yang, "NTRU Prime: NIST round 3 submission," Tech. Rep., 2020.
[48] S. Fluhrer, P. Kampanakis, D. McGrew, and V. Smyslov, *Mixing Preshared Keys in the Internet Key Exchange Protocol Version 2 (IKEv2) for Post-quantum Security*, document RFC 8784, Jun. 2020.
[49] S. Kousidis, F. Strenzke, and A. Wussler, *Post-Quantum Cryptography in OpenPGP*, document Internet-Draft draft-WUSSLER-OPENPGP-PQC-01, Internet Engineering Task Force, Work in Progress, Mar. 2023.

DUONG DINH TRAN received the B.S. degree in information technology from the VNU University of Engineering and Technology, in 2014, and the M.S. and Ph.D. degrees in information science from the Japan Advanced Institute of Science and Technology (JAIST), Japan, in 2020 and 2023, respectively. He is currently a Postdoctoral Researcher with JAIST. He has been working on applications of formal verification to different systems/protocols, such as concurrent/distributed systems and particularly, post-quantum cryptographic protocols.

KAZUHIRO OGATA received the B.S., M.S., and Ph.D. degrees in engineering from Keio University, in 1990, 1992, and 1995, respectively. He is currently a Professor with the Japan Advanced Institute of Science and Technology (JAIST). His research interest includes the applications of formal methods to systems, such as distributed systems and security protocols.

SANTIAGO ESCOBAR received the B.S., M.S., and Ph.D. degrees from Universitat Politècnica de València. He is currently a Full Professor with Universitat Politècnica de València. He is also a part of the development team of both the Maude high-performance modeling and programming language and the Maude-NPA cryptographic protocol analyzer. His research interests include formal methods, security, verification, model checking, rewriting, narrowing, and evaluation strategies.

SEDAT AKLEYLEK received the B.Sc. degree in mathematics major in computer science from Ege University, Izmir, Turkey, in 2004, and the M.Sc. and Ph.D. degrees in cryptography from Middle East Technical University, Ankara, Turkey, in 2008 and 2010, respectively. From 2014 to 2015, he was a Postdoctoral Researcher with the Cryptography and Computer Algebra Group, TU Darmstadt, Germany. From 2016 to 2022, he was an Associate Professor with the Department of Computer Engineering, Ondokuz Mayıs University, Samsun, Turkey. Since 2022, he has been a Professor with the Department of Computer Engineering, Ondokuz Mayıs University. Since 2022, he has also been with the Chair of Security and Theoretical Computer Science, University of Tartu, Tartu, Estonia. His research interests include the post-quantum cryptography, algorithms, and complexity, architectures for computations in finite fields, applied cryptography for cyber security, malware analysis, the IoT security, and avionics cyber security. He is an Editorial Board Member of IEEE Access, *Turkish Journal of Electrical Engineering and Computer Sciences*, *PeerJ Computer Science*, and *International Journal of Information Security Science*.

AYOUB OTMANI was a Research Scientist with Inria, Paris, France, from 2009 to 2011. He was an Associate Professor with the ENSICAEN and the University of Caen, France, from 2004 to 2012. He is currently a Professor with the University of Rouen Normandie, France. His research interests include coding theory, information theory, code-based cryptology (cryptography and cryptanalysis), algorithmic, and complexity issues in coding theory and cryptography.