Formal specification of the post-quantum signature scheme FALCON in Maude

Víctor García[1,*], Santiago Escobar[1] and Kazuhiro Ogata[2]

[1]Universitat Politècnica de València (UPV), Camí de Vera, s/n, 46022 València, Valencia, Spain
[2]Japan Advanced Institute of Science and Technology (JAIST), Ishikawa 923–1292, Japan

Abstract

Cryptography is an important piece in any communication system. Digital signatures are a part of cryptography that ensures the authenticity and integrity of digital assets, vital properties for any secure communication. Due to fast advances in post-quantum technologies, the National Institute of Standards and Technologies started the Post-Quantum Cryptography project to standardise new algorithms and protocols that are secure against quantum attackers. One of the finalists is the signature scheme FALCON. We present the first formal specification, in the high-performance language Maude, of FALCON. For this purpose, we use an existing framework, originally aimed to formally specify and analyse post quantum key encapsulation mechanism. With the infrastructure provided by the framework, we encode a symbolic model of the signature scheme's behaviour and simulate an execution trace.

Keywords

Formal specification, Post-Quantum, Signature scheme, FALCON, Maude

1. Introduction

Security is a basic requirement in any information system today, where cryptography has an important role in fulfilling such needs. A building block of cryptography is a digital signature, a cryptography element that provides authentication and integrity protection to the system where it is enforced. The security of the most used digital signature schemes, e.g. Rivest-Shamir-Adleman (RSA) [1] and Elliptic Curve Digital Signature Algorithm (ECDSA) [2], is based on the hardness of solving computational problems hard to solve for classic computers, such as the prime factorization and discrete logarithm problems. Some algorithms could provide solutions to these hard computational problems. One example is Shor's prime factorization algorithm [3] when operating on computers with quantum computing capabilities.

With the race for quantum supremacy at full throttle there is a need for new secure schemes and protocols that are quantum-resistant. The National Institute of Standards and Technology (NIST) launched the Post-Quantum Cryptography (PQC) project to encourage the development

FAVPQC 2023: 2nd International Workshop on Formal Analysis and Verification of Post-Quantum Cryptographic Protocols, November 21, 2023, Brisbane, Australia

*Corresponding author.
vicgarval@upv.es (V. García); sescobar@upv.es (S. Escobar); ogata@jaist.ac.jp (K. Ogata)
https://v1ct0r-byte.github.io/ (V. García); http://personales.upv.es/sanesro/ (S. Escobar); http://www.jaist.ac.jp/~ogata/ (K. Ogata)
0000-0003-0681-1130 (V. García); 0000-0002-3550-4781 (S. Escobar); 0000-0002-4441-3259 (K. Ogata)

of new protocols resilient to adversaries with access to quantum computing power. So far, at the time of writing this paper, there have been 4 rounds, where different Public-key Encryption and Key-establishment Algorithms (KEMs) and Digital Signature Algorithms (DSAs) were submitted for standardization. The first selection of finalists took place in round 3, where CRYSTALS-Kyber was the only selected candidate to represent the KEMs section. Moreover, CRYSTALS-Dilithium, FALCON and SPHINCS+ were the finalists for DSAs. From the three finalists in the DSAs section we focus on FALCON [4].

In this paper, we present the first steps towards modelling signature schemes by reusing and adapting a framework previously developed for Key Encapsulation Mechanisms in [5]. Specifically, we show how the framework is adapted to signature schemes and how FALCON is described in this new framework, obtaining an executable symbolic model that simulates execution traces. All the specifications presented in this paper can be found in the GitHub repository at: https://github.com/v1ct0r-byte/PQC-in-Maude/tree/PQ-SIG.

Outline: The rest of the paper is structured as follows. Section 2 lists a series of formal analysis tools/techniques over post-quantum protocols. Section 3 explains basic concepts on Maude, the framework we work on and the basic idea of signature schemes. Section 4 presents the signature scheme detailing the steps in each algorithm. Section 5 settles assumptions for our symbolic model and displays the Maude specification in greater detail, focusing on the specification of FALCON. Section 6 gives some concluding remarks and future work.

2. Related Work

Advances in protocol security analysis have been made in the field of cryptography. One interesting idea is the one proposed at [6], where the author explains several examples of formal specification of protocols and introduces and explains the symbolic and computational model analysis approaches. In [7], the authors explore the current literature and papers on both symbolic and computational analysis of protocols. In this survey, they analyze the results by combining both types of analysis. This proposal was initially made by [8] to close the gap between both lines of protocol verification.

Among the various symbolic protocol analysis tools available, we have Maude-NPA [9, 10], related to the programming language Maude [11, 12, 13]. Maude-NPA has a theoretical basis on rewriting logic, unification and narrowing and performs a backwards search from a final attack state to determine whether or not it is reachable from an initial state. Some symbolic tools, such as ProVerif [14, 15], are based on an abstract representation of a protocol using Horn clauses. The verification of security properties is done by reasoning on these representative clauses. Other symbolic tools, such as Tamarin [16, 17], are based on constraint solving to perform an exhaustive, symbolic search for execution traces. Furthermore, other symbolic tools such as Scyther [18] or CPSA [19] attempt to enumerate all the essential parts of the different possible executions of a protocol. Also, AKISS [20] or the DEEPSEC prover [21] are other tools mostly used to decide equivalence properties.

Some related work can be found for the symbolic security analysis of protocols with quantum features. For example, the authors of [22] build a model of IKEv2 on a classical setting and

then perform some analysis on it for seven properties, all of it using the Tamarin prover. With this first symbolic analysis, they prove their model to be correct and corroborate previous results by other authors. Later, they extended the model with the improvements included in the latest extension of IKEv2 in order to include a quantum-resistant key exchange. With this extension, they perform a new analysis, where all properties hold, verifying the security of the new extension.

Another paper performing symbolic analysis of post-quantum protocols is [23]. The authors use the recently published tool SAPIC$^+$ to analyze the Ephemeral Diffie Hellman Over COSE (EDHOC) protocol, on its 12th draft version. With SAPIC$^+$ they can automatically transform their model to a suitable one in Tamarin, ProVerif or DEEPSEC respective syntax. This allows the authors to take advantage of the strength of each tool to perform analysis over their modular composed model of the protocol. With the analysis, they discover several flaws and report them to the team of EDHOC. The authors proposed some fixes, validated them and the proposal was accepted into the 14th draft version of the protocol.

In [24] a variant for a handshake protocol from the WireGuard VPN protocol with post-quantum capabilities is presented. They perform such adaptation by replacing the previous Diffie-Hellman-based handshake with key-encapsulation mechanisms. The authors verify the security of their proposal with symbolic and computational proofs. On the one hand, the symbolic proofs verify more security properties than the computational proofs and are computer-verified. On the other hand, computational proofs give stronger security guarantees as the proof makes less idealizing assumptions.

The closest works to our contributions in this paper are [25, 26, 27, 28, 5]. [25] provides a first approximation on the symbolic specification of post-quantum protocols in Maude-NPA. The authors decided to specify the Post-Quantum TLS protocol primitives and execution trace. This type of work is interesting and necessary to us because it demonstrates the capability of Maude-NPA, and Maude, to verify more advanced schemes automatically. In the extended version, [26], the authors report some of the experimental results of formal verification/analysis over Transport Layer Security using a parallel version of Maude-NPA. [27, 28] present the first symbolic security analyses of a collection of post-quantum protocols. These analyses served as guidelines for the developments presented in [5]. The main differences of our paper with [5] are: (i) we focus on signature schemes, (ii) the three main algorithms are modified to represent the new steps and (iii) communication now takes place in only one way due to the nature of digital signature protocols.

Finally, regarding the post-quantum signature scheme FALCON, as far as the authors know there are no formal specifications or analyses in the literature.

3. Preliminaries

The section briefly introduces the language used to specify our model, Maude. Moreover, a high-level view of the framework used to specify the signature schemes is explained. Finally, an explanation of the nature and general behaviour of signature schemes is given.

3.1. Maude

Maude [12, 11] is a high-level programming language and system implementing rewriting logic [29]. Rewriting logic is ideally suited to specify and execute computational systems in a simple and natural way. Some examples are the works in Petri nets [30], process calculus [31], object-based systems [32], asynchronous hardware [33], a mobile ad hoc network protocol [34], cloud-based storage systems [35], web browsers [36], programming languages with threads [37], distributed control systems [38] and models of mammalian cell pathways [39, 40].

Rewriting logic has a sub-logic called membership equational logic. This sub-logic defines a system's deterministic parts using functional modules. In contrast, Maude system modules represent concurrent systems as conditional rewrite theories that model a nondeterministic system which may never terminate and where the notion of a computed value may be meaningless. In this concurrent system, the membership equational sub-theory defines the states of such a system as the elements of an algebraic data type, such as terms in an equivalence class associated with cryptography properties. We can call this aspect the static part of the specification. Instead, its dynamics, i.e., how states evolve, are described by the transition rules, which specify the possible local concurrent transitions of the system.

In the case of analysis, Maude provides a series of tools to analyze specified models. The most basic form of system analysis is illustrated by the `search` command in Maude. The command performs reachability analysis from an initial state to a target state, searching for states that violate the invariant. If the invariant fails to hold, it will do so for some finite sequence of transitions from the initial state, which will be uncovered by the `search` command above since all reachable states are explored in a breadth-first manner. If the invariant does hold, we may be lucky and have a finite state system, in which case the `search` command will report failure to find a violation of the invariant. However, the search will never terminate if an infinite number of states are reachable from the initial state. Moreover, under the assumption that the set of states reachable from an initial state is finite, Maude also supports explicit-state model checking verification of any properties in linear-time temporal logic (LTL) through its LTL model checker.

3.2. Framework

Our work is developed on the framework proposed from [5]. The framework's purpose is to ease the specification of Key Encapsulation Mechanism (KEMs) in a modular way, splitting the different components into modules. Moreover, with the specifications at hand, the framework allows the user to run analyses such as invariant analysis or model checking of properties. In [5], the authors present three case studies where Kyber, Classic McEliece and BIKE are modelled and analysed. These KEMs were candidates of the same round but under the Public-key Encryption and Key-establishment Algorithm section.

A general picture of the framework is given in Figure 1. There are different modules in the framework, depicted by the colours of the box, and inclusion relations between modules are represented by arrows. Functional modules are in blue, and system modules are in red. The data structures and main algorithms (KeyGen, Enc and Dec) are represented in functional modules through symbols and equations, e.g. `DATA-TYPES` module specifies the different data types and their properties the scheme handles. Functional module `MODEL-CONFIGURATION`

serves as a basis for the system module, defining the structure upon which the behaviour of the scheme will be specified later. The general configuration handles participants in the protocol, a network for message passing between the participants, and lists of contact symbols to be used as keys and values in the symbolic computations. Finally, everything blends in the system module KEM. There, the communication actions of honest participants and intruders are represented by transition rules between states of the general configuration. Thanks to the data and cryptography properties defined through an equational theory in the functional modules, symbolic simulations of the protocol can be performed.

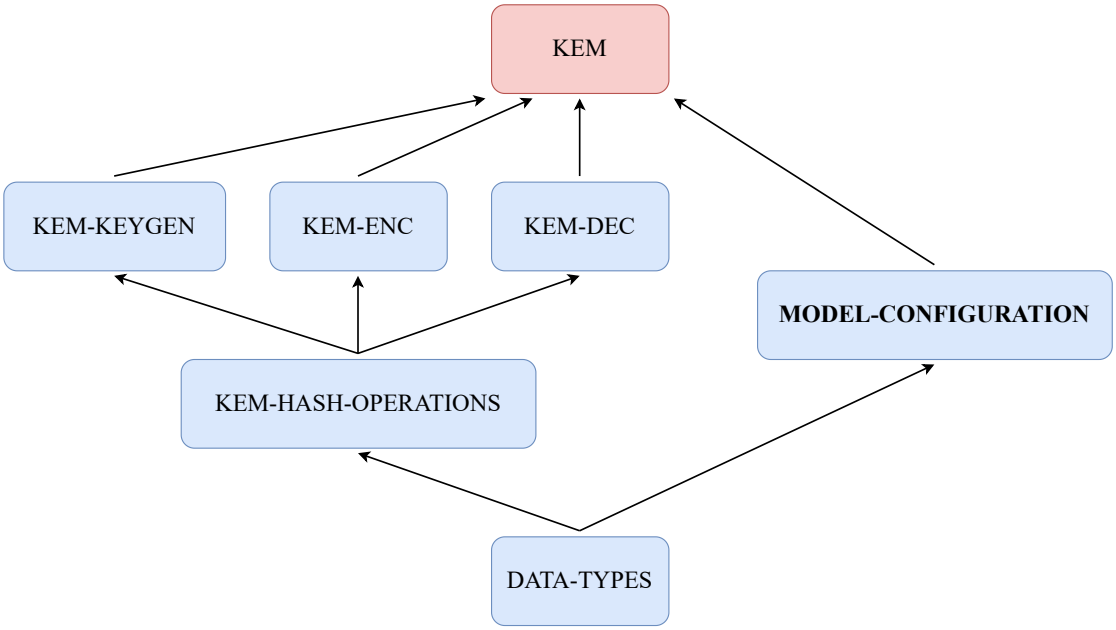

Figure 1: Overview of the framework, extracted from [5], used to specify and analyze post-quantum key encapsulation mechanisms in a modular way.

3.3. Signature schemes

A signature scheme is a mathematical scheme that uses asymmetric cryptography to verify the authenticity of digital assets, e.g. messages or documents. The asymmetric cryptography takes place in the form of a pair of keys, one public and one private. As an overview, the private key is used to sign the asset. Meanwhile, the public key, known by the other participant, is required to verify the validity of the generated signature. Signature schemes provide a mechanism for participants to verify that a message comes from someone they know, given that they know the public key of the sender.

Let's see an example of a general flow of a signature scheme. Figure 2 depicts a network diagram between two participants, Alice and Bob. The participants perform a series of actions depending on their role in the signature scheme. Let us suppose that Alice initiates the signature process since she wants to send a message, our digital asset, to Bob. To achieve this, Alice first needs to generate a pair of keys, a publicly known key and a secret key. After generating the

pair of keys, Alice uses the secret key over the message to generate a signature of the message. Moreover, since Alice wants to make sure Bob receives a valid version, she sends the signature along with the original message to Bob. Upon reception of the message and signature, Bob can validate the message's signature by using Alice's public key. If the signature proves valid, then Bob has strong guarantees that the message comes from Alice and thus knows the message is authentic and keeps its integrity.

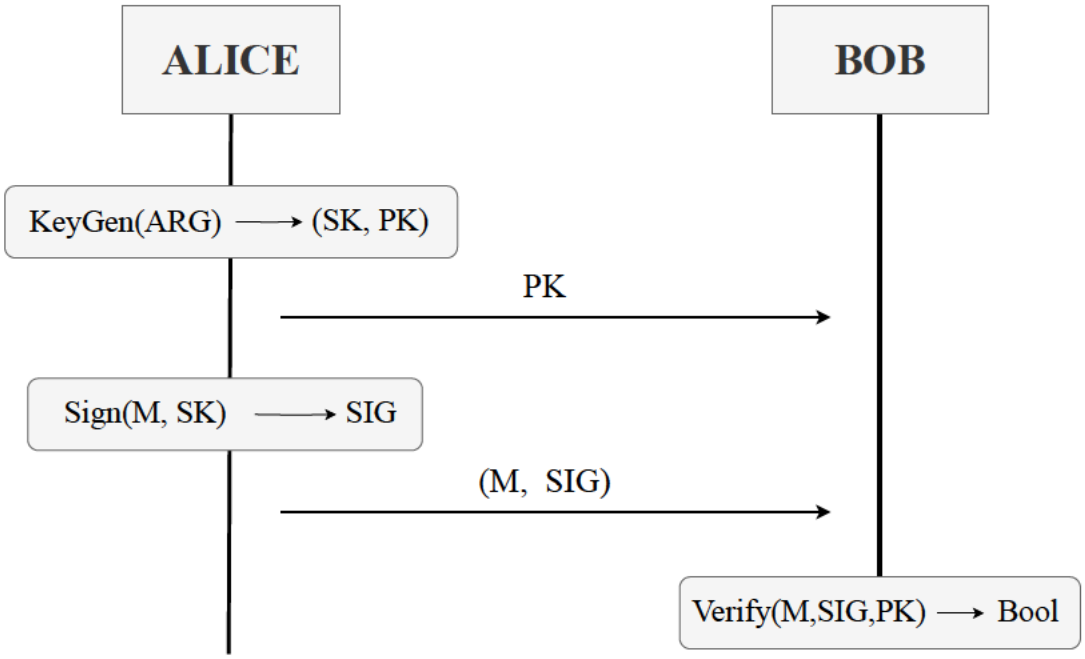

Figure 2: Network diagram of a signature scheme between Alice and Bob.

4. FALCON

Fast-Fourier Lattice-based Compact Signatures over NTRU [4] is a signature scheme based on lattices to sign and verify messages. The main components are a class of cryptography lattices, the NTRU class is chosen, and a trapdoor sampler, for which a new technique has been developed, the fast Fourier sampling. Concrete algorithms for each of the main three functions are presented in Figure 3, and will serve as guidelines to our Maude specifications.

The first algorithm, **KeyGen**, generates a pair of keys depending on two arguments: ϕ and q. These two arguments serve as seeds for the whole scheme and are required for the computation of polynomials f, g, F and G. These polynomials then form the components of a matrix used to compute what the authors call a FALCON tree. This tree, along with a normalized version of the matrix is what defines the secret key $sk = (\hat{B}, T)$. The public key is then given by the value $h = gf^{-1} \mod q$ in step 7 of the algorithm, where the first two polynomials are multiplied (one in inverse form) under the modulus of q, one of the arguments that serve as seed.

The algorithm to sign some digital assets, **Sign**, needs the digital asset in question and the private key. Note that, in principle, the algorithm would also receive a certain bound β, but we simplified the specification based on the assumptions given in Section 5.1. First, a random value r is uniformly sampled, to which the message to sign is concatenated and given to a hash operation to produce a value c. This hash value c is used along the components of the secret key to compute a pair of values s_1 and s_2. Finally, s_2 is compressed to serve as a component of the signature along with the random value r. Thus, the signature $sig = (r, s)$ is the pair of values of the uniformly sampled r and the compressed value s_2.

The verification algorithm, **Verify**, uses the sent asset along with its signature, together with the public key of the sender. Similar to the signing algorithm, a bound β is needed, but we omit it due to the assumptions given in Section 5.1. The verification procedure is as follows. The value c is recomputed using the received message m with the second component of the signature, i.e. the salt r. Then, values s_1 and s_2 are recomputed based on the recomputed value c. The value for s_2 is obtained through the decompression of the compressed value in s. Meanwhile, s_1 depends on the recomputed values of c and s_2, plus the public key, to be recomputed using the formula in step 3. If the recomputed values s_1 and s_2 match the original ones then the signature is proven to be valid for the received message m. Otherwise, the signature is rejected and the validity of the message can not be proven valid.

$$
\begin{array}{l}
\textbf{KeyGen} : (\phi, q) \rightarrow (sk, pk) \\
1 : f, g, F, G \leftarrow NTRUGen(\phi, q) \\
2 : B \leftarrow mat(f, g, F, G) \\
3 : \hat{B} \leftarrow FFT(B) \\
4 : G \leftarrow \hat{B} \times \hat{B}^* \\
5 : T \leftarrow ffLDL^*(G) \\
6 : sk \leftarrow (\hat{B}, T) \\
7 : h \leftarrow gf^{-1} \mod q \\
8 : pk \leftarrow h
\end{array}
$$

$$\xrightarrow{\quad pk \quad}$$

$$
\begin{array}{l}
\textbf{Sign} : (m, sk) \rightarrow sig \\
1 : r \leftarrow \{0,1\}^{320} \\
2 : c \leftarrow HashToPoint(r||m, q, n) \\
3 : t \leftarrow (FFT(c), FFT(0))\hat{B}^{-1} \\
4 : z \leftarrow ffSampling(t, T) \\
5 : s = (t - z)\hat{B} \\
6 : (s_1, s_2) \leftarrow invFFT(s) \\
7 : s \leftarrow Compress(s_2, 8 \cdot sbytelen - 328) \\
8 : sig = (r, s)
\end{array}
$$

$$\xrightarrow{\quad (m,\ sig) \quad}$$

$$
\begin{array}{l}
\textbf{Verify} : (m, sig, pk) \rightarrow Accept/Reject \\
1 : c \leftarrow HashToPoint(r||m, q, n) \\
2 : s_2 \leftarrow Decompress(s, 8 \cdot sbytelen - 328) \\
3 : s_1 \leftarrow c - s_2 h \mod q
\end{array}
$$

Figure 3: Falcon internal algorithms adapted from [4].

5. Formal specification with Maude

This section takes a closer look at the formal specification built in Maude. First, a series of assumptions over the model are made so we can abstract from computational requirements and focus on the symbolic behaviour, easing the specification. Then we explain how we used the existing framework to define the different rules, and how the equational theory represents the cryptography properties making the specification executable.

5.1. Assumptions

When symbolically modelling a protocol one can abstract himself from computational aspects to make the specifications simpler and easier to understand. Such abstractions come at a cost, making the model less representative of what really happens in the real world, but facilitating the reasoning of the intended behaviour. The assumptions we make for our symbolic specification over the computational model are:

- Any generated polynomials `f`, `g`, `F` and `G` always satisfy the equation $fG - gF = q \mod \phi$. This helps to encapsulate on the respective symbolic values any polynomial that will satisfy the equation.
- The signature and verification algorithms operate always in the bound given by their parameter β, thus it is not deemed necessary in the specification of these algorithms. The assumption allows us to avoid specifying the parameter in algorithms for signature generation and validation, easing the procedure. The main loss would be the symbolic model's fidelity over an actual implementation, but this can apply to most kinds of assumptions given their nature.
- The correctness of the encodings `s1` and `s2` in signature and validation algorithms. In other words, signature validation of a well-constructed signature for a valid message does not fail. This assumption allows us to focus on one of the outcomes of the validation algorithm, i.e. the case in which the signature is proven valid, skipping what would happen if the signature is rejected.
- Modulus operation over `q` is equal to modulus operation over `(q, ϕ)`. This is one of the most important assumptions we have made, since it will allow us to model the relation $s_1 + s_2 h = c \mod (\phi, q)$ to later obtain s_1 as the equivalent equality $c - s_2 h \mod q = s_1$.

5.2. Code specification

The framework from [5] serves as a perfect frame for specifying signature schemes. The main differences fall in (i) the participants' behaviour, since now messages go only one way through the communication channel, and (ii) the three main algorithms and their related modules. In the case of the communication, the rules used to send and receive messages are similar in both KEMs and DSAs. However, the main algorithms in digital signature schemes differ in content and application from those of KEMs, thus their modules hold new declarations to use in the redefined rules. We now dive into the details of each of the main algorithms of FALCON specified in Maude. Having in mind Figure 3 will help in the explanations.

5.2.1. KeyGen

Similar to KEMs, signature schemes require a key generation algorithm to provide a pair of keys. We have defined in module FALCON-KEYGEN, partially shown in Figure 4, the necessary symbolic values and operations for this algorithm. Specifically, we have defined operator constants phi and q to represent any possible value of ϕ and q respectively. The generator of polynomials *NTRUGen* is declared and defined through the equation to return a list of polynomials f, g, F and G, which are symbolic representations of any possible combination of them when receiving the constants phi and q as parameters. Furthermore, operations regarding Fast Fourier Transformation (FFT), the product between matrices and ffLDL have also been defined.

```
fmod FALCON-KEYGEN is
    ...
    --- Sample values for KeyGeneration
    op phi : -> Polynomial .
    op q : -> Nat .

    --- Sampler of values f, g, F and G
    op NTRUGen : Polynomial Nat -> List{Data} .
    ops f g F G : -> Polynomial .

    eq NTRUGen(phi, q) = (f g F G) .

    --- Matrix of a set of polynomials (this is to construct B)
    op mat : List{Data} -> Matrix .

    --- Fast Fourier Transformation
    op FFT : Matrix -> Polynomial .
    op FFT : Polynomial -> Polynomial .

    --- Matrix product
    op _x_ : Polynomial Polynomial -> Matrix .

    --- Fast Fourier LDL
    op ffLDL : Matrix -> Polynomial .
endfm
```

Figure 4: Functional module to declare and define values and operations to perform the key generation in FALCON.

The rule for key generation requires two samples from different groups of samples available as Figure 5 shows in the first line. Specifically, sample values for ϕ and q from the pools phis and qs are taken. These values serve as the basics for the construction of the public and secret keys. The conditions of the conditional rule KeyGen serve as representations for some of the internal steps in the algorithm. The first step is the generation of a list of polynomials L through function NTRUGen on the sampled values SAM1 and SAM2. Then, we collapse some of the specification

steps in the definition of the secret key SK. The secret key is defined in this second step as a pair of elements. The first element of the pair is the result of applying FFT over the resulting matrix from the elements of L, i.e. f, g, F and G. The second element of the pair is the product between ffLDL over the first element of the pair with that same element. Finally, the public key is defined as the product of the second element of L and the inverse of the first element of L, modulus the sampled value for q.

```
crl [KeyGen] :  {phis(SAM1, CONT1), qs(SAM2, CONT2), CONT3}
                < (ID1[emptyK]peer(ID2)) PS >
                net(MSGS)
                =>
                {phis(CONT1), qs(CONT2), CONT3}
                < (ID1[publicKey(ID1,PK) ; secretKey(ID1,SK)]
                   qI(ID1,SAM2), phiI(ID1,SAM1), peer(ID2)) PS >
                net(MSGS)
                if  L := NTRUGen(SAM1,SAM2) /\
                    SK := ([FFT(mat(L)),ffLDL(FFT(mat(L)) x FFT(mat(L)))]) /\
                    PK := (elem(2,L) p* inv(elem(1,L))) mod SAM2 /\
                    ID1 =/= ID2 .
```

Figure 5: Conditional rule for the specification of a participant applying the KeyGen algorithm.

5.2.2. Sign

Signature schemes require operations to sign the digital asset, in this case, a message. To this end, we defined a functional module named FALCON-SIGN, depicted in Figure 6. Similar to module FALCON-KEYGEN, the module stores the declarations of sample values necessary for the signature, as well as operations. Constant values are declared to represent the salt r and message m. Furthermore, operators s1 and s2 symbolically represent any value for s_1 and s_2 respectively. Regarding operations, the sampling procedure ffSampling and the inverse Fast Fourier Transformation invFFT are declared, plus the second one is also defined to return the pair of values s1 and s2. Finally, an equation representing the property $s1 + s2h = c \mod q$ is given. This will prove to be very useful when trying to validate the signature in the **Verify** algorithm.

Figure 7 depicts the rule that models the behaviour of a participant applying the Sign algorithm. The initial requirements for the rule to apply over a participant are: (i) there is some message to be created, (ii) there is a sample value for r available and (iii) the participant has a secret key and a value q obtained from the key generation algorithm. Then, the conditions of the rule specify how the components are defined following the specification of FALCON. The first step is to create a hash value c from the message STR and the salt r. In the next two steps, a preimage of c is stored in t and is given as a parameter to ffSampling along the secret key. The result is then put in a formula, following the pattern on the right-hand side of the equation for invFFT in Figure 6. This allows the final step to return the pair of short polynomials s1 and

```
fmod FALCON-SIGN is
    ...
    --- Sample values for r
    op r : -> Nat .

    --- Sample values for messages
    op m : -> String .

    --- Fast Fourier Sampling
    op ffSampling : Polynomial Polynomial -> Polynomial .

    --- Constant values to represent an instance of s_1 and s_2
    ops s1 s2 : -> Polynomial .

    --- Inverse Fast Fourier Transformation
    op invFFT : Polynomial -> Pair .
    eq invFFT((P1:Polynomial p- P2:Polynomial) p* FFT(M1:Matrix)) = ([s1,s2]) .

    --- Equation specifiyng equality "s1 + s2h = c mod (q)"
    eq (P1:Polynomial p- (s2 p* P2:Polynomial)) mod N:Nat = s1 .
endfm
```

Figure 6: Functional module to declare and define values and operations to sign a message in FALCON.

s2. Finally, a signature associated with STR is placed in the contents of the participant. This signature is composed of the sampled random salt r and the compressed value of *s2*. Take note that the second component is compressed with a value called SBYTELEN. This symbolic constant is defined to represent the value of the byte length of s.

```
crl [Sign] :    {ms(str(STR), CONT1), rs(SAM1, CONT2), CONT3}
                < (ID1[secretKey(ID1,SK) ; KS1]qI(ID1,Q), CONT4) PS >
                net(MSGS)
                =>
                {ms(CONT1), rs(CONT2), CONT3}
                < (ID1[KS1]sig(STR,SAM1,Compress(second(Ss),SBYTELEN)),
                          mI(ID1,STR), qI(ID1,Q), CONT4) PS >
                net(MSGS)
                if  c := HashToPoint(SAM1 || STR, Q, n) /\
                    t := [FFT(c), FFT(0)] p* inv(first(SK)) /\
                    z := ffSampling(t, second(SK)) /\
                    s := (t p- z) p* first(SK) /\
                    Ss := invFFT(s) .
```

Figure 7: Conditional rule for the specification of a participant applying the Sign algorithm.

5.2.3. Verify

The last algorithm in the scheme, Verify, is depicted with the conditional rule of the same name in Figure 8. The participant must be peers with another one and have received both a public key and a message along with its corresponding signature from that peer. Now, similar to the Signing algorithm a hash value `c` of the message is computed. As the condition shows, the random value `R` is obtained from the first component of the signature pair. Then, the recomputation of values `s2` and `s1` takes place. On the one hand, obtaining the value of `s2` is straightforward since `P = Compress(s2, SBYTELEN)`. Through the equation `Decompress(Compress(S,N),N) = S`, where `S` is some polynomial value and `N` is some numeric value, we obtain *s2*. On the other hand, the value of `s1` requires an equational theory based on a relation between values. The specific relation is given by the equality $s_1 + s_2 h = c \mod (\phi, q)$ available in Algorithm 10 in [4], which we translated into the equivalent equation $c - s_2 h \mod (\phi, q) = s_1$ by leaving s_1 on the right side alone. This equation makes explicit the relation between the hash c, the public key h and the pair of values s_1 and s_2. The last condition models the check that the computed values must be equal to those used during the signing, thus verifying the signature of the message.

```
crl [Verify] :  {CONT1}
                < (ID1[publicKey(ID2, p mod Q) ; KS1]
                   mI(ID2,STR), sig(STR,R,P), peer(ID2), CONT2) PS >net(MSGS)
                =>
                {CONT1}
                < (ID1[KS1]mI(ID2,STR), peer(none), CONT2) PS >
                net(MSGS)
                if  c := HashToPoint(R || STR, Q, n) /\
                    S2 := Decompress(P, SBYTELEN) /\
                    h := p mod Q /\
                    S1 := (c p- (S2 p* h)) mod Q /\
                    (S1 == s1) /\ (S2 == s2) .
```

Figure 8: Conditional rule for the specification of a participant applying the Verify algorithm.

5.2.4. Execution

To prove termination of our formal model we run the `rewrite` command to see if the initial state ends in a state where two honest participants, Alice and Bob, apply the specified rules. Figure 9 shows the result of applying the said command over initial state `init1`, which is defined as `phis(phi), qs(q), ms(str(m)), rs(r)< (Alice[emptyK]peer(Bob)) (Eve[emptyK]peer(none)) (Bob[emptyK]peer(Alice)) >net(emptyM)`, where from left to right it defines a configuration with: (1) a set of samples, each with one element available, (2) a set of participants with no keys and two of them set to begin the protocol between them, and (3) an empty network of messages, i.e. there is no previous history of message exchanges. When applying the command we get a new configuration where Alice and Bob finished the communication between them, thus they are no longer peers and Bob has the message Alice

wanted to send him. With it we have proven termination, a liveness property, of the model. We do not prove any other properties, and leave them for future work.

```
                    \|||||||||||||||||/
                  --- Welcome to Maude ---
                    /|||||||||||||||||\
           Maude 3.3.1 built: Apr 13 2023 16:09:10
            Copyright 1997-2023 SRI International
                  Fri Oct 20 12:49:40 2023
==========================================
rewrite in FALCON : init1 .
rewrites: 30 in 0ms cpu (0ms real) (240000 rewrites/second)
result [GlobalState]:
{phis(emptyC),qs(emptyC),ms(emptyC),rs(emptyC)}
<
(Alice[emptyK]peer(none),phiI(Alice, phi),qI(Alice, q))
(Eve[emptyK]peer(none))
Bob[emptyK]peer(none),mI(Alice, m)
>net(msg{(Alice,Bob)[received](g p* inv(f)) mod q}
       msg{(Alice,Bob)[received]str(m),[r,Compress(s2, SBYTELEN)]})
```

Figure 9: Execution of the rewrite command over an initial state `init1` to check if the symbolic specification represents an execution of the signature scheme FALCON.

6. Conclusion and future work

We provide a first approach on the formal specification of post-quantum signature schemes. Specifically we adapted the framework from [5] to serve as a tool for the specification of signature schemes. The post-quantum signature scheme FALCON served as our case study to show the feasibility of the Maude language to represent the behaviour of post-quantum signature schemes along with their cryptography properties.

This work serves as a first step towards the formal verification of signature schemes. Thus, as future work we plan to use explicit state model checking for the verification of invariants and LTL Model Checking to check if any interesting property holds in the specified symbolic model.

References

[1] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (1978) 120–126.

[2] W. J. Caelli, E. P. Dawson, S. A. Rea, Pki, elliptic curve cryptography, and digital signatures, Computers & Security 18 (1999) 47–66.

[3] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM review 41 (1999) 303–332.

[4] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, Z. Zhang, Falcon: Fast-fourier lattice-based compact signatures over ntru (2020).
[5] V. García, S. Escobar, K. Ogata, S. Akleylek, A. Otmani, Modelling and verification of post-quantum key encapsulation mechanisms using maude, PeerJ Computer Science 9 (2023) e1547.
[6] B. Blanchet, Security protocol verification: Symbolic and computational models, in: International Conference on Principles of Security and Trust, Springer, 2012, pp. 3–29.
[7] V. Cortier, S. Kremer, B. Warinschi, A survey of symbolic methods in computational analysis of cryptographic systems, Journal of Automated Reasoning 46 (2011) 225–259.
[8] M. Abadi, P. Rogaway, Reconciling two views of cryptography (the computational soundness of formal encryption), Journal of cryptology 15 (2002) 103–127.
[9] S. Escobar, C. Meadows, J. Meseguer, Maude-NPA: Cryptographic protocol analysis modulo equational properties, in: Foundations of Security Analysis and Design V, Springer, 2009, pp. 1–50.
[10] S. Escobar, C. Meadows, J. Meseguer, A rewriting-based inference system for the nrl protocol analyzer and its meta-logical properties, Theoretical Computer Science 367 (2006) 162–202.
[11] M. Clavel, F. Durán, S. Eker, S. Escobar, P. Lincoln, N. Martí-Oliet, J. Meseguer, R. Rubio, C. Talcott, Maude Manual (version 3.3.1), Technical Report, SRI International, 2023. URL: http://maude.cs.illinois.edu.
[12] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. M. Oliet, J. Meseguer, C. Talcott, All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic, Lecture Notes in Computer Science, Springer, 2007. URL: http://dx.doi.org/http://dx.doi.org/10.1007/978-3-540-71999-1. doi:10.1007/978-3-540-71999-1.
[13] F. Durán, S. Eker, S. Escobar, N. Martí-Oliet, J. Meseguer, R. Rubio, C. Talcott, Programming and symbolic computation in maude, Journal of Logical and Algebraic Methods in Programming 110 (2020) 100497.
[14] B. Blanchet, B. Smyth, V. Cheval, M. Sylvestre, Proverif 2.00: automatic cryptographic protocol verifier, user manual and tutorial, Version from (2018) 05–16.
[15] B. Blanchet, V. Cheval, V. Cortier, Proverif with lemmas, induction, fast subsumption, and much more, in: 2022 IEEE Symposium on Security and Privacy (SP), IEEE, 2022, pp. 69–86.
[16] S. Meier, B. Schmidt, C. Cremers, D. Basin, The tamarin prover for the symbolic analysis of security protocols, in: International conference on computer aided verification, Springer, 2013, pp. 696–701.
[17] D. Basin, C. Cremers, J. Dreier, R. Sasse, Tamarin: verification of large-scale, real-world, cryptographic protocols, IEEE Security & Privacy 20 (2022) 24–32.
[18] C. J. Cremers, The scyther tool: Verification, falsification, and analysis of security protocols, in: International conference on computer aided verification, Springer, 2008, pp. 414–418.
[19] J. Ramsdell, J. Guttman, CPSA4: A cryptographic protocol shapes analyzer, https://github.com/mitre/cpsaexp, 2018.
[20] I. Gazeau, S. Kremer, Automated analysis of equivalence properties for security protocols using else branches, in: Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings,

Part II 22, Springer, 2017, pp. 1–20.
[21] V. Cheval, S. Kremer, I. Rakotonirina, The DEEPSEC prover, in: International Conference on Computer Aided Verification, Springer, 2018, pp. 28–36.
[22] S.-L. Gazdag, S. Grundner-Culemann, T. Guggemos, T. Heider, D. Loebenberger, A formal analysis of IKEv2's post-quantum extension, in: Annual Computer Security Applications Conference, 2021, pp. 91–105.
[23] C. Jacomme, E. Klein, S. Kremer, M. Racouchot, A comprehensive, formal and automated analysis of the edhoc protocol, in: USENIX Security'23-32nd USENIX Security Symposium, 2023.
[24] A. Hülsing, K.-C. Ning, P. Schwabe, F. Weber, P. R. Zimmermann, Post-quantum wireguard, in: 2021 IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 304–321.
[25] D. D. Tran, C. M. Do, S. Escobar, K. Ogata, Hybrid post-quantum tls formal specification in maude-npa-toward its security analysis?, Proceedings http://ceur-ws.org ISSN 1613 (2022) 0073.
[26] D. D. Tran, C. M. Do, S. Escobar, K. Ogata, Hybrid post-quantum transport layer security formal analysis in maude-npa and its parallel version, PeerJ Computer Science 9 (2023) e1556.
[27] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani, Formal specification and model checking of lattice-based key encapsulation mechanisms in Maude, in: Rewriting Logic and its Applications 14th International Workshop, WRLA 2022, 2022, p. 26.
[28] D. D. Tran, K. Ogata, S. Escobar, S. Akleylek, A. Otmani, Formal specification and model checking of saber lattice-based key encapsulation mechanism in Maude, in: Proceedings of the 34th International Conference on Software Engineering and Knowledge Engineering, 2022.
[29] J. Meseguer, Conditional rewriting logic as a united model of concurrency, Theor. Comput. Sci. 96 (1992) 73–155. URL: https://doi.org/10.1016/0304-3975(92)90182-F. doi:10.1016/0304-3975(92)90182-F.
[30] M.-O. Stehr, J. Meseguer, P. C. Ölveczky, Rewriting logic as a unifying framework for petri nets, in: Unifying Petri Nets, Springer, 2001, pp. 250–303.
[31] N. Martí-Oliet, J. A. Verdejo-López, Implementing CCS in Maude, in: Actas de las VIII Jornadas de Concurrencia: Cuenca, 14 a 16 de junio de 2000, Universidad de Castilla-La Mancha, 2000, pp. 81–96.
[32] J. Meseguer, A logical theory of concurrent objects and its realization in the Maude language, in: G. Agha, P. Wegner, A. Yonezawa (Eds.), Research Directions in Concurrent Object-Oriented Programming, MIT Press, 1993, pp. 314–390.
[33] M. Katelman, S. Keller, J. Meseguer, Rewriting semantics of production rule sets, Journal of Logic and Algebraic Programming 81 (2012) 929–956.
[34] S. Liu, P. C. Ölveczky, J. Meseguer, Modeling and analyzing mobile ad hoc networks in Real-Time Maude, Journal of Logical and Algebraic Methods in Programming (2015).
[35] R. Bobba, J. Grov, I. Gupta, S. Liu, J. Meseguer, P. Ölveczky, S. Skeirik, Design, Formal Modeling, and Validation of Cloud Storage Systems using Maude, in: R. H. Campbell, C. A. Kamhoua, K. A. Kwiat (Eds.), Assured Cloud Computing, J. Wiley, 2018, pp. 10–48. URL: http://hdl.handle.net/2142/96274.
[36] S. Chen, J. Meseguer, R. Sasse, H. J. Wang, Y.-M. Wang, A systematic approach to uncover

security flaws in gui logic, in: 2007 IEEE Symposium on Security and Privacy (SP'07), IEEE, 2007, pp. 71–85.

[37] J. Meseguer, G. Roşu, The rewriting logic semantics project, Theoretical Computer Science 373 (2007) 213–237.

[38] K. Bae, J. Meseguer, P. C. Ölveczky, Formal patterns for multirate distributed real-time systems, Science of Computer Programming 91 (2014) 3–44.

[39] S. Eker, M. Knapp, K. Laderoute, P. Lincoln, J. Meseguer, K. Sonmez, Pathway logic: Symbolic analysis of biological signaling, in: Biocomputing 2002, World Scientific, 2001, pp. 400–412.

[40] C. Talcott, S. Eker, M. Knapp, P. Lincoln, K. Laderoute, Pathway logic modeling of protein functional domains in signal transduction, in: Biocomputing 2004, World Scientific, 2003, pp. 568–580.