Formal Analysis of Julia Key Agreement Protocol

Navya Sivaraman[1][0000−0003−0123−1970],
Simin Nadjm-Tehrani[1][0000−0002−1485−0802], and
Thomas Johansson[2][0000−0003−1798−570X]

[1] Linköping University, Sweden
{navya.sivaraman,simin.nadjm-tehrani}@liu.se
[2] Lund University, Sweden
thomas.johansson@eit.lth.se

Abstract. The evolution of the fifth-generation network (5G) increases the demand and use of Internet of Things (IoT) devices extensively. The increased number of IoT devices increases the possibility of new attack surfaces, and thus even resource-constrained IoT devices need secure communication. In this work, we consider the Julia Key Agreement (JKA) protocol, which has been proposed as a secure and efficient protocol for communication among resource-constrained IoT devices. We formally model two variants of the JKA protocol and verify the intended security requirements, such as mutual authentication, forward secrecy, backward secrecy, and resilience to key impersonation attacks, using the Tamarin prover. Our formal analysis shows that the JKA protocol is susceptible to replay attacks under the Dolev-Yao threat model. We also expand the threat model by including several strong threat assumptions to discover interesting attack vectors.

Keywords: Key Agreement protocol · IoT security · Formal verification · Tamarin.

1 Introduction

The evolution of the fifth-generation wireless network (5G) increases the widespread connectivity among a multiplicity of end user nodes with high speed and low latency. New Radio (NR) access features enable machine-to-machine, device-to-device, and device-to-everything communication, including the Internet of Things (IoT) and Internet of Vehicles [1]. This in turn facilitates the widespread usage of IoT technology across various domains, such as smart healthcare, smart cities, surveillance cameras, and so on.

While there are many well-established protocols in the security area (e.g., TLS), and several of them have been extensively analysed for satisfying the claimed properties [2], the new IoT landscape introduces new ones, where some are not yet well-known or highly utilised. For example, Ma et al. [3] propose a three-party authentication protocol to establish mutual authentication in Internet of Vehicular (IoV) networks. This protocol includes heavy computational operations (i.e., number of scalar multiplications upto 17) but still vulnerable to

ephemeral secret leakage attacks [4]. The question is though, should one deploy such protocols with resource hungry operations and let systems to be compromised before spending effort in their detailed analysis, or should one adopt a "secure-by-design" approach where a component that has massive deployment potential is first scrutinised for its critical properties?

In this work, we adopt the latter approach and consider the Julia Key Agreement (JKA) protocol, which has been proposed as a secure and efficient protocol for communication among resource-constrained IoT devices [5]. The protocol is interesting since it is designed to reduce both the time and energy spent for key agreement while providing the mutual agreement protocol. Hence, we consider this to be a good representative for a family of resource-efficient protocols. The efficiency comes from reducing the number of scalar multiplications that are used in the agreement protocol.

There are two variants of JKA protocol: JKA with single scalar multiplication and JKA with two scalar multiplications. The authors for the protocol conjecture that the protocol protects against "compromised-key impersonation". However, there are no formal proofs to show the strengths or weaknesses of the protocol.

A key agreement protocol requires several computational operations to satisfy important security properties. The elliptic curve cryptosystem, which performs asymmetric (or public-key) cryptography, includes a lot of scalar multiplication operations [7], which is a resource-hungry operation [5]. In mathematics, scalar multiplication is the multiplication of a scalar (integer) with a group element. A group consists of a set of distinct group elements associated with a binary operation. Here, the group we consider is the cryptographic group, which satisfies the criteria of elliptic-curve cryptography, such as Curve25519 [8], and NIST P-256 [9]. Most of the key agreement protocols require at least three scalar multiplications to satisfy the basic security requirements. JKA uses fewer multiplications, but we need to find a relevant approach to model it formally so that our proofs provide assurance of security properties, or their absence.

Formal verification is an effective method to find weaknesses and potential vulnerabilities early in the design phase of a protocol. It relies on mathematical methods to verify the correctness of protocol designs. With formal methods, it is possible to analyze several aspects of a protocol using different threat scenarios to check whether the protocol provides essential security guarantees in presence of different threat assumptions [6].

Contributions. To the best of our knowledge, we are the first to perform a comprehensive formal analysis of the JKA protocol. We begin with assuming a powerful threat model namely Dolev-Yao, and perform the security evaluation of the JKA protocol by formally verifying relevant security properties, such as mutual authentication, forward and backward secrecy, and check whether the protocol is resilient to key-compromise impersonation (KCI) attacks. In addition, we formally verify the key secrecy property for both the static secret key and the ephemeral secret key.

Based on formal modelling and analysis within the Tamarin prover [16], we show that the two variants of the JKA protocol are vulnerable to replay attacks.

These attacks break one of the most important properties of key agreement, i.e., mutual authentication. We further expand the threat model based on different threat assumptions to discover interesting attack vectors.

The adopted methodology is common for all works on formal verification of protocols, namely formal modelling of the protocol in the language of some tool that aids proofs, choosing and formally stating the relevant properties, deciding and formally representing various adversarial scenarios, and performing a proof in the selected tool.

2 Background

This section provides an overview of the Julia Key Agreement protocol and security protocol verification.

2.1 Julia Key Agreement Protocol

To establish secure communication between resource-constrained IoT devices, Lundberg and Feljan [5] introduce the Julia Key Agreement (JKA) protocol.

Let us consider two communication parties, P_1 and P_2, who communicate over a network with JKA protocol. The JKA protocol consists of three phases: (i) *Static key generation*, (ii) *Pre-handshake*, and (iii) *Handshake*.

- *Static key generation*: This is the initial phase of the JKA protocol in which both communication parties, P_1 and P_2, derive the long-term static secret keys. This long-term static key derivation is a single-time procedure in the lifetime of an IoT device whereby the two communication parties P_i generate the static key pair $< s_i, S_i >$, where s_i is the static secret private key and S_i is the static public key. The static private keys (s_i) remain secret for each communication party, whereas the static public keys (S_i) are known to both parties.
- *Pre-handshake*: This is the second phase of the key agreement protocol, and it includes the pre-computation operations for the session and storing of the computed secrets on the devices in advance by both communication parties, P_1 and P_2. Examples of pre-computed secrets include hashes or ephemeral keys computed during the Pre-handshake stage for a particular session.
- *Handshake*: The final phase of the JKA protocol is the *Handshake* phase. During the *Handshake* phase both communication parties exchange the pre-computed secrets and internally compute the common symmetric key. Finally, P_1 and P_2 use this symmetric key to encrypt/decrypt data.

These are the three phases of the JKA protocol described in detail in the original reference [5]. In this paper, we use the term long-term keys and static keys interchangeably.

Fig. 1 depicts both the version of JKA with two scalar multiplications (right) and the version with single scalar multiplication (left). The main difference between JKA with single scalar multiplication (JKA_1) and JKA with two scalar

multiplications (JKA_2) is that the former applies to communication parties who have pre-knowledge of whom they are communicating with, i.e., the communication party P_1 has pre-knowledge about the intended receiver, P_2. Note that Fig. 1 excludes the pre-knowledge phase for the identity of receivers in JKA_1 (which is not discussed in the original paper either), and the *Static key generation* phase for both versions since our main goal here is to focus on secure operation of the *Handshake* and its preceding phase.

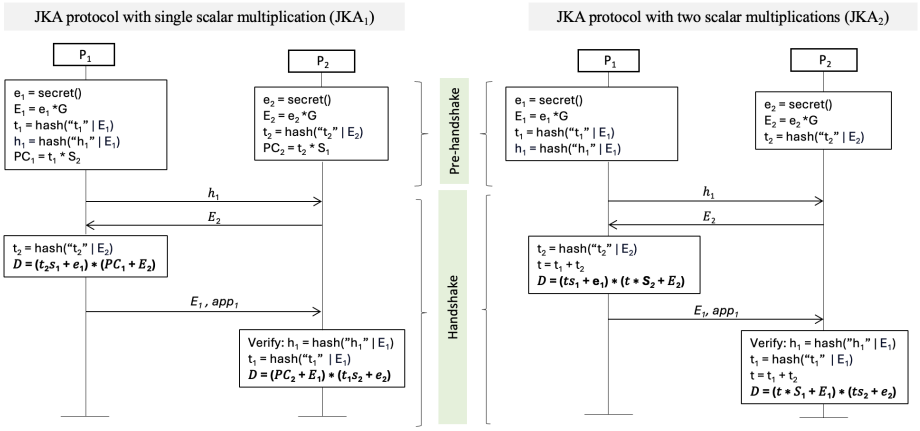

Fig. 1. JKA protocol with single scalar and two scalar multiplications.

During the *Pre-handshake* phase of JKA_2, P_1 and P_2 generate an ephemeral private key e_i using the *secret* function. This function returns a new random secret key in the form of a scalar, and this scalar acts as the input for the scalar multiplication in the additive group. The scalar multiplication of private ephemeral secret key e_i with the group generator G computes the ephemeral public key E_i. Here, the * symbol denotes the scalar multiplication. The ephemeral secrets are only for one-time usage in a session and are discarded after that. The *hash function* takes the string, representing a parameter ("t_i", and "h_i"), and the public ephemeral key (E_i) to calculate the hashes, i.e, t_i, and h_i, in the form of a scalar. Here, the parameters enclosed within double quotes represent a string, and the pipe symbol (|) denotes string concatenation. The two strings, "t_i" and "h_i", are unique and different.

When the communication parties, P_1 and P_2, complete the computation of the secrets during the *Pre-handshake* phase, the *Handshake* phase begins. During the *Handshake* phase, P_1 sends the pre-computed hash h_1 to P_2, and P_2 shares the ephemeral public key E_2 with P_1. The communication party P_1, who receives the ephemeral public key E_2, computes the hash t_2 using the *hash* function (as shown in Fig. 1).

After that, both communication parties (P_1 and P_2) of the JKA_2 protocol compute the hashes and Julia secret (D). The communication party P_1 of JKA_2

computes the hashes $''t''$, and $''t_2''$, and the Julia secret D. In particular, P_1 computes the Julia secret D by the scalar multiplication (*) of the two terms enclosed within brackets. The sub-term within the first bracket, ts_1 is computed by the integer multiplication of hash $''t''$ with the static private key of P_1, which is s_1. The symbol + denotes the concatenation of the term ts_1 with ephemeral private key of P_1, i.e., e_1. The second sub-term consists of two terms, $t * S_2$ and E_2. The term $t * S_2$ denotes the scalar multiplication of the hash t with the static public key of P_2, S_2. Finally, the scalar multiplication of these two sub-terms computes the Julia secret D.

After computing D, P_1 uses D to encrypt the first message (app_1), and sends to P_2, along with its ephemeral public key E_1. When P_2 receives the first message from P_1, it verifies the hash h_1, and computes t_1, t, and D. P_2 can decrypt the message (app_1), sent by P_1, if and only if it computes the same Julia secret D. Note that the computation of Julia secret D requires two scalar multiplications.

The same type of operations are used somewhat differently in the case of JKA_1 (as shown in the left hand side of Fig. 1). In JKA_1, during the *Pre-handshake* phase, the communication parties, P_1 and P_2 compute the same secrets as in JKA_2 except the PC_i parameter. The two communication parties compute this additional parameter using the scalar multiplication (*) of the hash t_i with the static public key S_i. The computation of Julia secret D requires single scalar multiplication during the *Handshake* phase.

2.2 Security Protocol Verification

There are two fundamental formal modeling approaches used for security protocol analysis and verification, computational and symbolic modeling [34].

In computational modeling, the messages are bitstrings, cryptographic algorithms are functions over bitstrings, and the adversary represents any probabilistic polynomial-time Turing machine [10–12]. On the other hand, in symbolic modeling, cryptographic primitives are represented as function symbols, messages are terms over these primitives [13, 14].

In this work, we use the Tamarin prover, which is a state-of-the-art verification tool designed to analyze security protocols symbolically [15–17]. The default underlying threat model for the Tamarin prover is Dolev-Yao, a strong threat model with a powerful adversary who will be able to intercept and tamper with any publicly available information [13]. We explain the format and semantics of the Tamarin rules interleaved with the explanation of our model of the JKA protocol.

2.3 Security Requirements

The security requirements for JKA protocols are explicitly stated in the original work [5]. That paper mentions four critical security requirements for the security evaluation of JKA protocol, i.e., mutual authentication, forward secrecy, backward secrecy, and resilience to key-compromise impersonation attacks.

- **Mutual Authentication:** This is an important property in every key agreement protocol. Mutual authentication, also known as bidirectional or two-way authentication, defines a security procedure in which both parties involved in a communication must verify their identities for each other [18].
- **Forward Secrecy:** Forward secrecy, also known as perfect forward secrecy (sometimes called break-backward protection), is one of the strong secrecy properties of the key agreement protocols. A protocol satisfies the forward secrecy property if and only if the compromise of long-term keys does not compromise past session keys [19].
- **Backward Secrecy:** Backward secrecy, also known as future secrecy, ensures that an adversary cannot compromise future information with the compromise of long-term keys [21, 20].
- **Resilience to Key compromise impersonation attacks:** A protocol satisfies KCI resilience if the compromise of the long-term secret key of protocol participant A does not allow the adversary to impersonate an honest participant B to A [23, 22].

3 RELATED WORK

Various studies propose novel key agreement protocols to secure communication among resource-constrained IoTs and formally verify the security of the proposed protocols.

Jarosz et al. propose a new authentication and key exchange protocol, Lightweight Authentication and Key Exchange Protocol for Federated IoT, based on a distributed ledger for IoT devices operating in federated learning environments [24]. They formally verify the security properties of the protocol, such as secrecy, authentication, and freshness, using the formal tools Verifpal and the Tamarin prover.

Krishnasrija et al. propose a transitive device authentication protocol based on the Chebyshev polynomial [25]. This protocol aims to establish secure and fast communication among IoT devices in large and complex IoT systems like the smart city or smart industry by establishing a secure session key. They use BAN logic and Scyther to perform formal analysis and verify the security properties of the protocol, such as mutual authentication, perfect forward secrecy, and secrecy.

Ding et al. propose a new key synchronization update algorithm (TXLKS) and a new secure communication protocol suite (LCPT) for resource-constrained IoT devices and environments [26]. They use the Tamarin Prover to evaluate their security properties, such as authentication, secrecy, and forward security.

Mirsaraei et al. introduce a new three-factor-based authentication scheme for secure communication in IoT environments using the blockchain platform [27]. The protocol implements the Elliptic-Curve Cryptography algorithm and is formally verified in terms of its security properties, such as mutual authentication, secrecy, and forward security, using the AVISPA tool.

Li extends *Mirsaraei et al.*'s work and proposes a new three-factor authentication mechanism that guarantees complete forward secrecy and resistance to

key compromise impersonation attacks [28]. The Real-or-Random model and the Proverif tool is used to perform formal analysis and verification.

While the above works show active research on formally verifying device authentication protocols of various kinds, to the best of our knowledge, we are the first to formally analyze and verify the two variants of the JKA protocol [5]. Our work benefits from the formalisation style and patterns adopted in Tamarin for similar properties.

4 System Model and Threat Model

This section introduces the relevant actors within the JKA protocol system model we construct in this work. Our threat model reflects the adversary capabilities within the modeled scenarios.

4.1 System Model

The system model consists of a pair of IoT devices that can communicate with each other, e.g. in a smart home, or a device that can connect to the IoT server over an underlying network.

Another instantiation would be IoT devices with sensors and actuators connected to an edge server, representing an example of IoT device-to-server communication, e.g. a surveillance camera connected to a gateway.

The JKA protocol between the communication parties P_1 and P_2 supports key agreement with either the IoT device-to-device communication or the IoT device-to-server communication. In JKA_2, the communication parties, P_1 and P_2, denote the actors that perform IoT device-to-device communication. The JKA_1 scenario is intended for the case where the communication parties P_1 and P_2 have pre-knowledge about each other and suits the IoT device-to-server actors.

The static secret keys are pre-distributed and securely stored within each communication device. The static secret keys are unique for each communication device, and thus, access to them is limited to the device in which they are stored, while the static public keys are publicly visible for the communication devices within the network. The original JKA protocol also includes some pre-shared secrets (hashes) and ephemeral secrets that are stored or computed individually by each communication device and shared with the other communication device. How these secret strings are shared is not specified in the protocol. So we assume here that these are shared through a separate communication channel. The ephemeral secret pairs are freshly generated and unique for each session during communication. Hence, the ephemeral secrets are temporarily stored in the communication device for each session and discarded afterward.

4.2 Threat Model

For both protocol models, JKA_1 and JKA_2, we consider the powerful Dolev-Yao adversary as our threat model. The Dolev-Yao adversary can intercept, replay, send or delete any publicly available messages within the underlying network.

In order to formalize different threat scenarios, we extend the Dolev-Yao threat model with additional adversary capabilities and allow the adversary to perform the following actions:

- compromise the static secrets of the communication parties.
- compromise the ephemeral secrets of the communication parties.
- compromise the hashes, h_1, t_i.
- compromise the communication channel between the devices.

Since the IoT device deployment is in large scale and their configuration and maintenance are hard tasks, the adversary may possess the capability to compromise or gain access to the sensitive information storage of less secure IoT devices (due to the presence of vulnerabilities, such as misconfigurations [35], weak encryption [37], etc.). Hence, we assume that the adversary has the privilege to access sensitive information, such as static keys or ephemeral keys, stored in a less secure IoT device, or can compromise the communication channel.

5 Formal Modeling and Security Evaluation

Formal verification approaches aim to identify weaknesses in cryptographic protocols early in the design process, which helps to avoid security breaches. Formal analysis of a protocol requires resolving three sub-problems: how to model a protocol, the threat model or adversary knowledge, and security properties. The formulations may vary depending on the choice of the proof tool that will be used.

In our approach, we use the Tamarin prover, one of the most prominent security protocol verification tools, for the formal analysis of our protocol model. In this section, after a brief description of the Tamarin tool, we describe how to model the protocol, adversary capabilities, and security properties, and conclude by summarizing our results.

5.1 Modelling Assumptions

In our modeling approach, we initially assume that the communication parties, P_1 and P_2 of the two variants of the JKA protocol are *honest*, i.e., the communication parties that work as intended according to the protocol and uncompromised are considered as *honest*. Therefore, the static and ephemeral private keys generated by the *honest* communication parties remain secret.

The default attack model in the Tamarin prover is Dolev-Yao, the adversary has access to the same set of functions as the honest parties. This means that a leaked key gives (when a device is assumed to be compromised) the adversary the same capabilities as its owner.

5.2 Protocol Modelling

In the Tamarin prover, the protocols and the adversary knowledge are modeled using a multi-set rewriting rule. The security properties are defined using first-order logic and they are denoted by lemmas [15].

Rules operate on the system's state, which is represented using a multiset (i.e., a bag) of facts. The rules are comprised of premises (left-hand side facts), action facts, and conclusions (right-hand side facts). The rules are used to define a transition system, which maintains its global state as a set of facts. The initial state of the transition system is the empty multiset. Execution of rules shows the system's state transition from the premises, a set of facts, to the conclusions.

The rules in the Tamarin prover for protocol modeling are represented as follows:

```
[p] --[a]-> [c]
```

where `p` is the premise, `a` is the action fact, and `c` is the conclusion.

A rule can be executed only if all the facts on its premise (left-hand side) are available in the current state. During the rule execution, it consumes the facts in the premise (left-hand side), i.e. removing them from the global state, and produces facts in the conclusion (right-hand side), i.e. adding them to the global state.

The actions during protocol state transitions (rule execution) are captured by action facts that appear in the traces. The trace of a system or protocol is a sequence of events that starts from an empty state and consists of a sequence of labeled actions, captured by the action facts for a sequence of rules that were applied.

Modeling the Keys

This section explains how we model the static key pair and ephemeral key pair of the communication parties (P_1, and P_2) in the JKA protocol. We use a common modelling approach to model the keys in both variants of JKA protocol.

We exemplify a rule to generate the long-term key pairs for the communication party P_1 in our Tamarin model as follows:

```
rule Generate_key:
  let
    pkS1 = pmult(~s1, 'g')
  in
  [ Fr(~s1) ]
  --[ Init($P1), Honest($P1) ]->
  [ !Ltk($P1, ~s1), !Pk($P1, pkS1), Out(pkS1) ]
```

The rule `Generate_key` represents the initialization of the communication party P_1 along with it's static key pair generation. The rule `Generate_key` is comprised of several facts: `Fr(...)` is a fact in premises, `Init(...)` and `Honest(...)` denote action facts, and `!Ltk(...)`, `!Pk(...)` and `Out(...)` denote fact within conclusions.

`Fr(...)` fact is a built-in fact in Tamarin prover that represents a freshly generated name, here the generated static private key. The symbol $ indicates that the fact type `P1`, which denotes the identity of the communication party P_1 is public. The symbol ! with a fact (`!Ltk(...)`) indicates that the fact is persistent. Here, the fact `!Ltk(...)` (Long-term key) denotes the association of each communication party with its (long-term) static private key. Each fact contains one or more terms, such as `($P1, ~s1)`. The symbol `~` denotes the freshness of the terms. The rule begins with a `let-in` binding of Tamarin, which is used to present macros. Macros are the sub-terms which appear multiple times within a rule. The variable `pkS1` denotes the static public key of `P1`. The special facts, `In()` and `Out()` are used to send and receive messages over the network, modeling information which can be intercepted by the adversary. The variable `g` enclosed within single quotes is a constant in Tamarin, which denotes the group generator, and the function `pmult(...)` denotes scalar multiplication operation [15]. The `pmult(...)` is a function symbol from the `bilinear-pairing` theory, which is an extension of `diffie-hellman` theory. Therefore, this theory helps to model the cryptographic group that satisfies the protocol requirement.

The rule `Generate_key` initialize a communication party with a public identifier `P1`, and associates it with a freshly generated public private key pair, i.e., `(~s1)` and `pkS1`. The facts (`!Ltk($P1, ~s1)`) and (`!Pk($P1, pks1)`), associate the communication party `P1` with its private key `(~s1)` and public key (`pkS1`).

In our modeling approach, the ephemeral key pair and the hash during the *Pre-handshake* phase of the JKA protocol are modeled as follows:

```
rule Prehandshake_P1:
  let
    EphK1 = pmult(~eph1, 'g')
    h1  = h(<'h1', EphK1>)
    t1 = h(<'t1', EphK1>)
  in
  [ !Ltk( $P1, ~s1 ), Fr(~eph1) ]
--[   Honest($P1), StaticSecret($P1, ~s1)
 , EphSecret($P1, ~eph1), Pre_handshake_P1($P1, h1)]->
 [ !Ephk($P1, ~eph1), Out(EphK1), P1_prehandshake(<$P1, $P2, h1>) ...]
```

The rule `Prehandshake_P1` is a code fragment of the actual rule from our Tamarin model of JKA_2. This code fragment presents the *Pre-handshake* phase of the communication party P_1.

The facts `(~eph1)` and `(EphK1)` denote the private and public ephemeral keys of `P1`.

The variable `h1` denotes the hash generated by the communication party P_1 during the *Pre-handshake* phase. In Tamarin, the function symbol `h()` denotes a unary cryptographic hash function, and here it takes the concatenation of string `'h1'` and `EphK1` as input. The enclosure in <...> denotes concatenation in Tamarin.

The `StaticSecret(...)` is an action fact that denotes the secrecy of the static private key `(~s1)` of the communication party `$P1`. The action fact

`EphSecret(...)` denotes the secrecy of ephemeral private key `(~EphK1)` of `$P1`. The action fact `Pre_handshake_P1($P1, h1)` denotes the generation of hash `h1` during the *Pre-handshake* phase by the communication party (P_1) with an identity `$P1`. The fact `!Ephk($P1, ~eph1)` is a persistent fact which associates the communication party `$P1` with its private ephemeral key `(~eph1)`. We assume that the public ephemeral key is publicly available information, and thus we use `Out(EphK1)` fact here.

The rule `Prehandshake_P1` shows how we model the ephemeral secrets and hash, for the communication party P_1 during the *Pre-handshake* phase of the JKA_2 protocol in Tamarin. Similarly, we have the rule `Prehandshake_P2` to model the secrets of communication party P_2 during the *Pre-handshake* phase.

To model the JKA_1 protocol, we define a private function (`shared_key($P2)`), based on identity to obtain the pre-knowledge to the sender about the intended receiver. We use a similar approach in modeling the *Pre-handshake* phase of the JKA_1 protocol. In addition to the facts in `let-in` binding of the rule `Prehandshake_P1`, we add the variable `PC1 = pmult(t1, pkS2)` in the rule. Similarly we add `PC2 = pmult(t2, pkS1)` in the `Prehandshake_P2` rule of JKA_1 protocol.

Modeling the Adversary Knowledge

In addition to the default Dolev-Yao model, we define a set of rules to integrate different adversary capabilities in Tamarin.

```
rule Reveal_ltk:
  [ !Ltk( $P1, ~s1) ] --[ RevealLtk($P1) ]-> [ Out( ~s1) ]
```

The rule `Reveal_ltk` states that if an adversary obtains access to the device with the identifier `$P1`, then he/she will be able to reveal the private static key `(~s1)` using the action fact `RevealLtk(...)`. As a result, the private static key `(~s1)` is considered as sent out to the public channel, which the Dolev-Yao adversary has access to.

```
rule Ephemeral_Reveal:
  [ !Ephk( $P1, ~eph1 ) ] --[ RevealEphk( $P1 ) ]-> [ Out( ~eph1 ) ]
```

The rule `Ephemeral_Reveal` states that an adversary gains access to the ephemeral private key `(~eph1)` and reveals it to the public channel (`Out(~eph1)`) if he/she compromises the communication party `$P1`.

We use a similar approach to integrate the remaining threat assumptions (as listed in Section 4.2) into our Tamarin model.

Modeling the Julia Secret

The novel part and core of the Julia protocol is the Julia secret. We use the

expanded equation (Equation (1)) of Julia secret in our modeling approach, and this is common for both variants of the JKA protocol.

$$D = t^2 s_1 s_2 * G + t s_1 e_2 * G + t e_1 s_2 * G + e_1 e_2 * G \tag{1}$$

Modeling Julia secret involves a combination of bilinear maps, hashes, and complex mathematical operations. We define our own user-defined functions and equational theories to model the calculation of the Julia secret. The construction of user-defined equational theory ensures the validity and integrity of the computations within Julia secret expression (1).

Here a code fragment is shown to define different user-defined functions and equations to model Julia secret in our Tamarin model in the case of JKA_2 protocol.

```
functions: add/4, multp/2, Calc_D_P/1, D/0, dec/2, enc/2
equations:
          Calc_D_P(
                                add(multp(
                                          multp(
                                                multp(t,t),s1),s2),
                                multp(multp(t,s1),e2),
                                multp(multp(t,e1),s2),
                                multp(e1,e2))) = D,
          dec(enc(msg, key), key) = msg
```

We define the partial equation for Julia secret (`D`) using the function `Calc_D_P`. The scalar multiplication (*) with `G` is captured in a rule (`Handshake_P1`) later. The function `add(...)` denotes addition, and `multp(...)` denotes multiplication.

The function models the invocation of the calculation within the protocol during the *Handshake* phase, followed by the message exchange using the Julia secret as the `key` and the encryption of (app_1) within the message (instantiating `msg`). We define the functions, `enc(...)` and `dec(...)`, to perform encryption and decryption on the message (app_1) exchanged between the communication parties.

Tamarin restricts the use of user-defined functions and built-in functions (`pmult(...)`) together in a single equational theory. Therefore, we use the built-in `pmult(...)` in the *Handshake* phase to complete the Julia secret formulation as follows:

```
rule Handshake_P1:
  let
    D = pmult(Calc_D_P(
                           add(multp(multp(multp(t,t),~s1),~s2),
                           multp(multp(t,~s1),~eph2),
                           multp(multp(t,~eph1),~s2),
                           multp(~eph1,~eph2))), 'g')
    handshake_message = enc('app1',D)
  in
  [ ...]
```

```
--[  ...
    , Handshake_message_P1($P1, 'app1')
    , Handshake_P1_send($P1, EphK1, handshake_message)
    , JuliaSecret(D) ...]-> [..., Out(handshake_message) ]
```

The rule `Handshake_P1` presents the *Handshake* phase of the communication party P_1 during the JKA_2 protocol. Here, we take the `Calc_D_P(...)` and `'g'` as input to the `pmult(...)` to compute the final Julia secret (`D`). The variable `handshake_message` includes the encrypted message using the Julia secret (`D`) as `key`.

5.3 Formalizing security properties

We formally define the security requirements of JKA protocol properties as lemmas in the Tamarin specification language. Here, we explain six different lemmas within the model to prove the desired security requirements of the JKA protocol.

Executability lemma: The executability lemma aims to perform a sort of sanity check on the protocol model.

```
lemma JKA_executable:
  exists-trace
  " Ex #i1 #j1 #i2 #i3 #i4 #j2 P1 P2 h e2 e1 emsg dmsg.
      Init(P1)@i1 & Pre_handshake_P1(P1, h)@i2
      & Pre_handshake_P2(P2, e2)@j1
      & Handshake_message_P1(P1, dmsg)@i4
      & Handshake_P1_send(P1, e1, emsg)@i3
      & Decrypt_message(P2, dmsg)@j2
      & #i1 < #i2  & #i2 < #j1
      & #j1 < #i3  & #i3 < #j2 "
```

The lemma holds when there exists at least one trace in which the protocol performs the key agreement as expected. The action fact `Handshake_message_P1` denotes the message (`'app1'`) of communication party P_1 before encryption at a time point #i4, i.e., `dmsg`, and the action fact `Decrypt_message(...)` denotes that the communication party P_2 decrypts and obtains the same message `dmsg` at a later time point #j2. This implies that there exists at least one trace that satisfies the requirement of deriving the same Julia secret (`D`) by both communication parties `P1` and `P2`.

Injective Agreement: Injective agreement, also simply known as "agreement", is a key security requirement that needs to be accomplished among the communicating entities during a key agreement. We use Lowe's definition to formally define injective agreement [36].

Let's consider two entities, A (initiator) and B (responder), who act according to protocol with an agreement based on a set of data (for example, the

secret derived by both parties or a secret key or set of messages). Informally, the protocol satisfies the injective agreement property whenever the initiator A completes a run of the protocol with B as respondent, and B has been running the protocol with A prior to the completion. Also, both A and B should agree on the set of data considered during the agreement. Injective agreement satisfies the one-to-one relationship between A and B, i.e., A and B agree on the same session. The intention is to eliminate replay attacks.

For mutual authentication, we formulate two injective agreement lemmas; `injectiveagreement_P1_P2`, and `injectiveagreement_P2_P1`. Here, we only present the lemma `injectiveagreement_P1_P2`. We adapted the pre-defined lemma in the Tamarin manual [15] with respect to our protocol model.

```
lemma injectiveagreement_P1_P2:
  " All a b r #i. Commit(a,b,<'P1','P2',<'D',r>>)@i
      ==> (Ex #j. Running(b, a, <'P1','P2',<'D',r>>)@j & j < i
           & not (Ex a2 b2 #i2. Commit(a2, b2, <'P1','P2', <'D',r>>)@i2
                    & not (i2 = i)) ) "
```

In the lemma `injectiveagreement_P1_P2`, `a` and `b` can be considered as the communication parties `P1` and `P2` in JKA, and the variable `r` denotes the julia secret `D` in the JKA. The lemma states that for all instances in which `P1` commits to `P2` with the secret `D` at a time point #i, there exists a unique run of the protocol by `P2` with the same secret `D` at an earlier time point #j.

We use the action facts `Commit(...)` and `Running(...)` to formulate the injective agreement. These action facts follow a form in which the first argument describes the name of the acting agent, i.e., our `P1`, and the second argument is the name of the intended recipient, `P2`. The action fact `Commit(...)` is semantically considered as a claim made by `P1`, and expects a corresponding `Running(...)` action at `P2`.

Forward Secrecy: We formulate one of the strong secrecy properties, forward secrecy, within the context of the JKA as follows:

```
lemma forwardSecrecy:
  " All p d #i. JuliaSecret(p,d) @i
        & not (Ex #r. LtkReveal(p)@r & Honest(p) @i & r < i)
        ==> not (Ex #j. K(d)@j) "
```

The lemma `forwardSecrecy` states that for all (`All`) possible values of `p` (denote communication parties), and `d` (Julia secret) where the communication party `p` generates a secret `d` at a time point `#i`, given that the adversary has not revealed the long-term static private keys of the honest communication party `p` at any time point `#r` before (`i`) then the adversary cannot derive the secret (`d`).

Here, the Tamarin built-in function `K()` is used to denote that the adversary has some knowledge, specifically knowledge of the Julia secret `d`. Note that we use `Honest(p)` to denote that the communication party `p` is *honest*. As mentioned in

Section 5.1, a communication party `p` is *honest* when its keys (long-term static private key) remain uncompromised.

Backward Secrecy: In addition to forward secrecy, we formally define the backward secrecy lemma, which guarantees future security.

```
lemma backwardSecrecy:
  " All p d #i. JuliaSecret(p,d) @i
        & not (Ex #r. LtkReveal(p)@r & Honest(p) @i & i < r)
        ==> not (Ex #j. K(d)@j) "
```

The lemma `backwardSecrecy` states that for all possible values of `p` (denote communication parties), `d` (Julia secret) at a time point #i: the communication party `p` generates `d`, which is a secret, and the adversary has not revealed the long-term private keys of the honest communication party (`p`) after the Julia secret `d` is kept as secret (`i < r`), then the adversary cannot derive the secret (`d`).

Similar lemmas can be formulated for formally stating that static key secrecy and ephemeral key secrecy properties are maintained in the absence of a corresponding revealing.

Resilience to KCI: We reason about the resilience of the protocol in presence of KCI attacks by analyzing the mutual authentication and static key secrecy requirements. A counterexample will then show that the protocol is not resilient if a static secret key is revealed.

5.4 Analysis and Results

In this section, an overview of the analysis results is provided, followed by a detailed description of the specific scenario outcomes. We formally verify the security properties, injective agreement (authentication), forward secrecy, backward secrecy, and resilience to KCI attacks.

Since we observe similar results for both variants of the JKA protocol, we will hereafter refer to JKA for the outcomes. Table 1 summarizes the results from the formal analysis of the protocol. The first column in Table 1 presents our different threat assumptions. The rest of the columns present the above-mentioned security requirements.

The verification is started with the Dolev-Yao threat model, then assumes a compromise of the long-term static private keys of the communication parties, a compromise of the ephemeral private keys of the communication parties, a compromise of the hashes (h_1, t_i), and a compromise of the communication channel between the communication parties, i.e., the adversary can intercept all the exchanges in the communications during the *Pre-handshake* and *Handshake* phases.

Table 1. An overview of the security evaluation of JKA protocol. The symbol ✓ indicates the satisfiablity of the security requirement and ✗ denotes the presence of a counterexample.

Threat Scenario	Injective Agreement	Forward Secrecy	Backward Secrecy	Resilience to KCI Attacks
Dolev-Yao model	✗	✓	✓	✓
Static secret key reveal	✗	✓	✓	✗
Ephemeral key reveal	✗	✓	✓	✓
Reveal hashes h_1, t_i	✗	✓	✓	✓
Insecure communication channel	✗	✓	✓	✓

The first threat scenario includes the formal analysis of the JKA protocol in the presence of Dolev-Yao threat model. The proof shows a counterexample for the injective agreement lemma due to the presence of replay attacks during *Pre-handshake* and *Handshake* phases. During the *Pre-handshake* and *Handshake* phases, both communication parties exchange their public ephemeral keys to each other. An adversary who can eavesdrop on the publicly available ephemeral keys, can easily reuse it and continuously replay it to the other communication parties. This breaks the actual key agreement between the communication parties because the adversary acts as a legitimate communication party to the other one, and the communication party who receives these key values fails to distinguish the source (i.e., a genuine communication party or an adversary). Thus, authenticity is not guaranteed for the JKA protocol. However, the protocol satisfies the remaining security requirements, such as secrecy properties and resilience to KCI attacks.

The second threat scenario is the compromise of static secrets of both communication parties. This may be possible by the partial device storage compromise or a key leakage from a less secure IoT device. The Tamarin prover shows the presence of attack traces (counterexamples) for the injective agreement, static key secrecy, and resilience to KCI attacks. On the other hand, the JKA protocol satisfies the forward secrecy, backward secrecy, and ephemeral key secrecy requirements.

Next, we examine the threat scenario in which the adversary performs the ephemeral private key reveal for both communication parties. In this scenario, the Tamarin prover shows the presence of attack traces for the injective agreement and ephemeral key secrecy. Therefore, the adversary fails to derive the Julia secret even if the adversary compromises the ephemeral key of both communication parties. Thus, the JKA protocol satisfies the static key secrecy, forward and backward secrecy requirements. The JKA protocol is resilient to KCI attacks in this scenario.

Another threat scenario we consider here is the compromise of hashes, such as h_1, t_i. In this scenario, the adversary is able to perform replay attacks during the *Pre-handshake* phase of the JKA protocol, where the communication party

P_1 exchanges the hash h_1 with the other communication party P_2. However, the communication parties have not exchanged the other hashes, t_1, t_2, and t, during the JKA protocol communication. The Tamarin prover shows the presence of attack traces only for the injective agreement property. In this scenario, the protocol satisfies forward secrecy, backward secrecy, static key secrecy, ephemeral key secrecy, and resilience to KCI attacks.

Finally, we include a strong threat scenario by assuming the compromise of the communication channel between the two communication parties during protocol execution. The attacker can eavesdrop and learn the hash h_1 and ephemeral public key E_2 of the communication party P_2 during the *Pre-handshake* phase. Similarly, the attacker can eavesdrop on an encrypted message, i.e., encryption of message app_1 with Julia secret D, as well as the ephemeral public key E_1 from the communication party P_1 during the *Handshake* phase. Thus, this allows the adversary to perform replay attacks during the *Pre-handshake* and *Handshake* phases. This is confirmed in our proofs, where the Tamarin prover shows a counterexample for the injective agreement property. On the other hand, the protocol satisfies the security properties, forward secrecy, backward secrecy, static key secrecy, ephemeral key secrecy, and resilience to KCI attacks.

From our formal proofs and analysis, we can conclude that the JKA protocol is vulnerable to replay attacks. This breaks the mutual authentication property, which is the core security requirement of a key agreement protocol. However, the JKA protocol satisfies the forward and backward secrecy security requirements in five different threat scenarios. Therefore, the adversary fails to derive Julia secret, which is the core of JKA protocol, even in the presence of Dolev-Yao threat model, compromise the long-term static private keys of the communication parties, compromise the ephemeral private keys of the communication parties, compromise the hashes (h_1, t_i), and compromise the communication channel between the devices.

6 Conclusions and Future work

This paper presents a formal model of the JKA protocol for both variants from scratch using the symbolic modeling tool, Tamarin Prover. The formal verification of the security requirements of the JKA protocol, such as mutual authentication, forward secrecy, and backward secrecy are presented in detail. In particular, five threat scenarios are studied in depth for each variant (ten models in total). We examine the resilience of the JKA protocol to KCI attacks which corresponds to absence of impersonation in presence of a static key reveal in all scenarios. Our formal analysis of the JKA protocol shows that the protocol is susceptible to replay attacks. When the static key secrecy is violated the KCI resilience breaks. We have made responsible disclosure to the authors of the original paper to enable discussing their remedial action or comments in the final version of the paper.

With our formal verification results, it is evident that the JKA protocol is not secure for critical applications in presence of certain threat models. Our

comprehensive analysis suggests that the benefit from the small computation footprint of the protocol does not come together with the achievement of the intended security requirements.

While the invention of the Julia secret is a good step towards a secure communication between IoT devices, this work shows that a systematic analysis of all threat vectors may reveal unforeseen aspects during design. The analysis did require considerable effort but it is worthwhile since such protocols may become prevalent on a large scale. Our work demonstrates how to formally model any complex IoT protocol and verify its security requirements before real-world deployment.

Future work can extend the JKA protocol by proposing a solution to mitigate the replay attacks during protocol execution, such as using unique timestamps for each message exchange, and MAC to verify the integrity. The work in this paper can be a ground on which to build more complex scenarios with multi-protocol attacks. For example, JKA in combination with 5G (and beyond) would be a relevant goal. One could also study whether a combination of JKA_1 and JKA_2 or a switching between them under certain threat assumptions based on risk analysis is security-wise interesting.

Acknowledgments. This work was funded by ELLIIT, Excellence Center at Linkoping-Lund on Information Technology. The second author was also supported by the AIR^2 project partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. We thank Martin Gunnarsson for discussions about early drafts of this work.

References

1. Bockelmann, C., Pratas, N., Nikopour, H., Au, K., Svensson, T., Stefanovic, C., Popovski, P. and Dekorsy, A., 2016. Massive machine-type communications in 5G: Physical and MAC-layer solutions. IEEE communications magazine, 54(9), (pp.59-65). https://doi.org/10.1109/MCOM.2016.7565189
2. Cremers, C., Horvat, M., Hoyland, J., Scott, S. and van der Merwe, T., 2017. A comprehensive symbolic analysis of TLS 1.3. In ACM SIGSAC conference on computer and communications security, (pp. 1773-1788). https://doi.org/10.1145/3133956.3134063
3. Ma, M., He, D., Wang, H., Kumar, N. and Choo, K.K.R., 2019. An efficient and provably secure authenticated key agreement protocol for fog-based vehicular ad-hoc networks. IEEE Internet of Things Journal, 6(5), (pp.8065-8075). https://doi.org/10.1109/JIOT.2019.2902840
4. Ali, H. and Ahmed, I., 2024. LAAKA: Lightweight Anonymous Authentication and Key Agreement Scheme for Secure Fog-Driven IoT Systems. Computers & Security, 140, (p.103770). https://doi.org/10.1016/j.cose.2024.103770
5. Lundberg, F. and Feljan, J., 2021. Julia: fast and secure key agreement for IoT devices. In 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks, (pp. 90-99). https://doi.org/10.1145/3448300.3468116
6. Hofer-Schmitz, K. and Stojanović, B., 2020. Towards formal verification of IoT protocols: A Review. Computer Networks, 174, (p.107233). https://doi.org/10.1016/j.comnet.2020.107233

7. Ansari, B. and Hasan, M.A., 2008. High-performance architecture of elliptic curve scalar multiplication. IEEE Transactions on Computers, 57(11), (pp.1443-1453). https://doi.org/10.1109/TC.2008.133
8. Bernstein, D.J., 2006. Curve25519: new Diffie-Hellman speed records. In Public Key Cryptography-PKC: 9th International Conference on Theory and Practice in Public-Key Cryptography, (pp. 207-228). https://doi.org/10.1007/11745853_14
9. PUB, FIPS., "Digital signature standard (DSS).", Fips pub (2000): 186-192.
10. Goldwasser, S., Micali, S. and Rivest, R.L., 1988. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on computing, 17(2), (pp.281-308). https://doi.org/10.1137/0217017
11. Gordon, A.D. and Jefrey, A., 2002. Typing one-to-one and one-to-many correspondences in security protocols. In International Symposium on Software Security, (pp. 263-282).https://doi.org/10.1007/3-540-36532-X_17
12. Tedeschi, P., Sciancalepore, S., Eliyan, A. and Di Pietro, R., 2019. LiKe: Lightweight certificateless key agreement for secure IoT communications. IEEE Internet of Things Journal, 7(1), (pp.621-638). https://doi.org/10.1109/JIOT.2019.2953549
13. Dolev, D. and Yao, A., 1983. On the security of public key protocols. IEEE Transactions on information theory, 29(2), (pp.198-208). https://doi.org/10.1109/TIT.1983.1056650
14. Needham, R.M. and Schroeder, M.D., 1978. Using encryption for authentication in large networks of computers. Communications of the ACM, 21(12), (pp.993-999). https://doi.org/10.1145/359657.359659
15. D Basin, C Cremers, J Dreier, S Meier, R Sasse, and B Schmidt. 2016 [online]. Tamarin-Prover Manual Security Protocol Analysis in the Symbolic Model. https://github.com/tamarin-prover/tamarin-prover
16. Meier, S., Schmidt, B., Cremers, C. and Basin, D., 2013. The TAMARIN prover for the symbolic analysis of security protocols. In Computer Aided Verification (CAV): 25th International Conference, (pp. 696-701). https://doi.org/10.1007/978-3-642-39799-8_48
17. Schmidt, B., Meier, S., Cremers, C., and Basin, D., 2012. Automated analysis of Diffie-Hellman protocols and advanced security properties. In IEEE 25th Computer Security Foundations Symposium (pp. 78-94). https://doi.org/10.1109/CSF.2012.25
18. Otway, D. and Rees, O., 1987. Efficient and timely mutual authentication. ACM SIGOPS Operating Systems Review, 21(1), (pp.8-10). https://doi.org/10.1145/24592.24594
19. Menezes, A.J., Van Oorschot, P.C. and Vanstone, S.A., 2018. Handbook of applied cryptography. CRC press.
20. Unger, N., Dechand, S., Bonneau, J., Fahl, S., Perl, H., Goldberg, I. and Smith, M., 2015. SoK: secure messaging. In IEEE Symposium on Security and Privacy, (pp. 232-249). https://doi.org/10.1109/SP.2015.22
21. Yu, S.J., Lee, Y.C., Lin, L.H. and Yang, C.H., 2022. An Energy-Efficient Double Ratchet Cryptographic Processor With Backward Secrecy for IoT Devices. IEEE Journal of Solid-State Circuits. https://doi.org/10.1109/JSSC.2022.3220838
22. Chalkias, K., Baldimtsi, F., Hristu-Varsakelis, D. and Stephanides, G., 2009. Two types of key-compromise impersonation attacks against one-pass key establishment protocols. In E-business and Telecommunications: 4th International Conference, (pp. 227-238). https://doi.org/10.1007/978-3-540-88653-2_17
23. Gorantla, M.C., Boyd, C., Nieto, J.M.G. and Manulis, M., 2008. Modeling key compromise impersonation attacks on group key exchange protocols. ACM

Transactions on Information and System Security (TISSEC), 14(4), (pp.1-24). https://doi.org/10.1145/2043628.2043629
24. Jarosz, M., Wrona, K. and Zieliński, Z., 2022. Formal verification of security properties of the Lightweight Authentication and Key Exchange Protocol for Federated IoT devices. 17th Conference on Computer Science and Intelligence Systems (FedCSIS) (pp. 617-625). https://doi.org/10.15439/2022F169
25. Krishnasrija, R., Mandal, A.K. and Cortesi, A., 2023. A lightweight mutual and transitive authentication mechanism for IoT network. Ad Hoc Networks, 138, (p.103003). https://doi.org/10.1016/j.adhoc.2022.103003
26. Ding, Z., He, D., Qiao, Q., Li, X., Gao, Y., Chan, S. and Choo, K.K.R., 2023. A lightweight and secure communication protocol for the IoT environment. IEEE Transactions on Dependable and Secure Computing. https://doi.org/10.1109/TDSC.2023.3267979
27. Mirsaraei, A.G., Barati, A. and Barati, H., 2022. A secure three-factor authentication scheme for IoT environments. Journal of Parallel and Distributed Computing, 169, (pp.87-105). https://doi.org/10.1016/j.jpdc.2022.06.011
28. Li, Y., 2023. A secure and efficient three-factor authentication protocol for IoT environments. Journal of Parallel and Distributed Computing, 179, (p.104714). https://doi.org/10.1016/j.jpdc.2023.104714
29. Schiller, E., Aidoo, A., Fuhrer, J., Stahl, J., Ziörjen, M. and Stiller, B., 2022. Landscape of IoT security. Computer Science Review, 44, (p.100467). https://doi.org/10.1016/j.cosrev.2022.100467
30. Lin, X., Li, J., Baldemair, R., Cheng, J.F.T., Parkvall, S., Larsson, D.C., Koorapaty, H., Frenne, M., Falahati, S., Grovlen, A. and Werner, K., 2019. 5G new radio: Unveiling the essentials of the next generation wireless access technology. IEEE Communications Standards Magazine, 3(3), (pp.30-37). https://doi.org/10.1109/MCOMSTD.001.1800036
31. Chettri, L. and Bera, R., 2019. A comprehensive survey on Internet of Things (IoT) toward 5G wireless systems. IEEE Internet of Things Journal, 7(1), (pp.16-32). https://doi.org/10.1109/JIOT.2019.2948888
32. Sharma, S., Satapathy, S., Singh, S., Sahu, A.K., Obaidat, M.S., Saxena, S. and Puthal, D., 2018. Secure authentication protocol for 5G enabled IoT network. In Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC), (pp. 621-626). https://doi.org/10.1109/PDGC.2018.8745799
33. Norrman, K., Sundararajan, V. and Bruni, A., 2020. Formal analysis of EDHOC key establishment for constrained IoT devices. arXiv preprint arXiv:2007.11427. https://doi.org/10.48550/arXiv.2007.11427
34. Blanchet, B., 2012. Security protocol verification: Symbolic and computational models. In International Conference on Principles of Security and Trust, (pp. 3-29). https://doi.org/10.1007/978-3-642-28641-4_2
35. Bakhshi, T., Ghita, B. and Kuzminykh, I., 2024. A Review of IoT Firmware Vulnerabilities and Auditing Techniques. Sensors, 24(2), (p.708). https://doi.org/10.3390/s24020708
36. Lowe, G., 1997. A hierarchy of authentication specifications. In Proceedings of the 10th computer security foundations workshop, (pp. 31-43). https://doi.org/10.1109/CSFW.1997.596782
37. Alomari, A. and Kumar, S.A., 2024. Securing IoT Systems in a Post-Quantum Environment: Vulnerabilities, Attacks, and Possible Solutions. Internet of Things, (p.101132). https://doi.org/10.1016/j.iot.2024.101132