Journal of Visual Language and Computing

journal homepage: www.ksiresearch.org/jvlc

Graphical Animations of the NS(L)PK Authentication Protocols[⋆,⋆⋆]

Thet Wai Mon[a], Dang Duy Bui[a], Duong Dinh Tran[a], Canh Minh Do[a] and Kazuhiro Ogata[a,*]

[a]School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), 1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan

ARTICLE INFO

Article History:
Submitted 3.1.2021
Revised 6.1.2021
Second Revision 8.1.2021
Accepted 10.10.2021

Keywords:
graphical animation
NSPK authentication protocol
NSLPK authentication protocol
state machine
state picture design

ABSTRACT

NSLPK is visualized using SMGA so that human users can visually perceive non-trivial characteristics of the protocol by observing graphical animations. NSPK is a public-key authentication protocol invented by Needham and Schroeder and NSLPK is a revised version of NSPK by Lowe. These characteristics could be used as lemmas to formally verify that NSLPK enjoys desired properties. We first carefully make a state picture design for NSLPK to produce good graphical animations with SMGA and then find out non-trivial characteristics of the protocol by observing its graphical animations. Finally, we also confirm the guessed characteristics using model checking. The work demonstrates that SMGA can be applied to a wider class of systems/protocols, authentication protocols in particular. The visualization of NSLPK is different from ordinary message sequence diagrams that have been often used for security protocols. It is convenient that message sequence diagrams can be automatically generated in a graphically animated way for some cases such that we need to see the order in which way what messages are sent, faked and/or received. Thus, we have revised SMGA so that message sequence diagrams can be automatically generated in a graphically animated way.

1. Introduction

Authentication protocols have become important technical components in this advanced highly networked world. If authentication protocols have some flaws (security holes), users' credentials may be leaked to malicious third parties. It is then really important to make sure that authentication protocols are reliable and truly secure. Therefore, we need to use some technologies for this purpose. One possible technology is formal verification with theorem proving in which one challenging task is lemma conjecture. If human users carefully observe graphical animations of a state machine, they could recognize the characteristics from which they could conjecture useful lemmas. We aim to come up with a better way to conjecture lemmas in much fewer efforts and less time by observing animations produced by the State Machine Graphical Animation tool (SMGA) to complete formal proof.

SMGA [11] has been developed to visualize graphical animations of state machines that can be used to formalize security protocols. The main purpose of SMGA is to help human users be able to visually perceive non-trivial characteristics of state machines by observing their graphical animations because humans are good at visual perception [7]. SMGA takes a finite state sequence of a state machine formalizing a protocol and produces its graphical animation by regarding the state sequence as a movie film. Observing such a graphical animation allows us to guess the characteristics of the state machine. We confirm whether the state machine enjoys guessed characteristics because such characteristics may or may not be true properties of the state machine. One possible way to do so is model checking. However, it does not guarantee that the state machine enjoys the properties when the reachable state space is unbounded. If that is the case, we should use some other techniques, such as theorem proving to make sure that the system enjoys the guessed properties. Several case studies of some protocols have been

⋆This work was partially supported by JST SICORP Grant Number JPMJSC20C2, Japan and FY2020 grant-in-aid for new technology research activities at universities (SHIBUYA SCIENCE CULTURE AND SPORTS FOUNDATION).

⋆⋆The present paper is an extended and revised version of the paper [9] presented at DMSVIVA 2021.

*Corresponding author

thetwaimon@jaist.ac.jp (T.W. Mon); bddang@jaist.ac.jp (D.D. Bui); duongtd@jaist.ac.jp (D.D. Tran); canhdominh@jaist.ac.jp (C.M. Do); ogata@jaist.ac.jp (K. Ogata)

ORCID(s): 0000-0002-2700-1762 (D.D. Bui); 0000-0001-7092-2084 (D.D. Tran); 0000-0002-1601-4584 (C.M. Do); 0000-0002-4441-3259 (K. Ogata)

DOI reference number: 10.18293/JVLC2021-N2-005

conducted with SMGA, such as Qlock [1] and MCS [12, 3], shared-memory mutual exclusion protocols, Suzuki-Kasami protocol [2], a distributed mutual exclusion protocol, and ABP [11], a communication protocol. Authentication protocols have not been yet, and this work is the very first time of tackling authentication protocols with SMGA.

Authentication protocols are often visualized as message sequence diagrams called the Alice-Bob format, where Alice and Bob are principals. It is possible to grasp the messages exchanged by Alice and Bob and the order of the messages sent and received by principals and faked by Cathy, an intruder when such protocols are visualized as message sequence diagrams. We need to take into account the existence of intruders so as to formally verify that authentication protocols enjoy desired properties, such as the nonce secrecy property. Cathy plays an ordinary principal from the Alice and Bob point of view but does something against authentication protocols, such as faking messages based on the gleaned information. Thus, many messages may be faked by Cathy. If many messages appear, it may not be straightforward to comprehend message sequence diagrams. This is why we came up with a different way to visualize NSLPK than message sequence diagrams.

We aim at coming up with a brand-new way to visualize the behavior of authentication protocols. Since it is known that state picture designs affect how well human users can detect non-trivial characteristics of protocols [3], we carefully make a state picture design of the NSLPK protocol and based on it to produce good graphical animations. By observing the graphical animations, some non-trivial characteristics are guessed and checked with Maude [4]. In the paper, we mainly focus on how to design the state picture of the NSLPK protocol and how some characteristics could be found by observing graphical animations with detailed experiments.

However, it is convenient that message sequence diagrams can be automatically generated in a graphically animated way for some cases such that we need to see the order in which way what messages are sent, faked and/or received. Thus, we have implemented SMGA-SD that is a tool that automatically generates message sequence diagrams in an animated way from a finite sequence of states. We have integrated SMGA-SD with SMGA. We have visualized NSPK and NSLPK in SMGA-SD.

The remaining part of the paper is organized as follows. Sect. 2 gives some preliminaries such as state machine, Maude, and SMGA. Sect. 3 describes the NSLPK protocol and Sect. 4 describes its formal specification. Sect. 5 presents the state picture design of the NSLPK protocol in which the idea and the design are mainly conveyed. Sect. 6 reports on how we can find characteristics by observing graphical animations. Sect. 7 describes SMGA-SD and some experiments with SMGA-SD. Sect. 8 concludes the paper with some pieces of future work.

2. Preliminaries

This section describes some preliminaries needed to comprehend what follows in the present paper: state machines, Maude, SMGA, NSLPK protocol. State machines are mathematical models used to formalize systems. Maude is a rewriting specification/programming language in which state machines can be described. Maude also refers to its processor equipped with model checking functionality. NSLPK protocol is an authentication protocol and used as one main example in the present paper.

2.1. State machines

A state machine is a mathematical model of computation. Based on the current state and given input, state machine performs state transitions and produces outputs. A state machine $M \triangleq \langle S, I, T \rangle$ consists of a set S of states, a set $I \subseteq S$ of initial states, and a binary relation $T \subseteq S \times S$ over states. $(s, s') \in T$ is called a state transition where s' is successor state of s and may be written as $s \rightarrow_M s'$. The set $R \subseteq S$ of reachable states with respect to (wrt) M is inductively defined as follows: $I \subseteq R$ and if $s \in R$ and $s \rightarrow_M s'$, then $s' \in R$. A state predicate p is an invariant property wrt M if and only if $(\forall s \in R)\ p(s)$ that is $p(s)$ holds for all $s \in R$. A state predicate p can be interpreted as a set P of states such that $(\forall s \in P)\ p(s)$ and $(\forall s \notin P)\ \neg p(s)$. A finite sequence $s_0, \ldots, s_i, s_{i+1}, \ldots, s_n$ of states is called a finite computation of M if $s_0 \in I$ and $(s_i, s_{i+1}) \in T$ for each $i = 0, \ldots, n-1$.

Systems can be formalized as state machines. States are expressed as braced soups of observable components. Soups are associative-commutative collections, and observable components are name-value pairs. That is a state of S is expressed as associative-commutative collection of name-value pairs. The juxtaposition operator is used as the constructor of soups. Let $oc1, oc2, oc3$ be observable components, and $oc1\ oc2\ oc3$ is the soup of observable components. Then a state that consists of these three observable components can be expressed as $\{oc1\ oc2\ oc3\}$, which equals $\{oc3\ oc1\ oc2\}$ and some others due to associativity and commutativity. To specify state transitions we use Maude as a formal specification language.

Let us consider the hand game 'Rock Paper Scissors' between a human (you) and a machine and a system (called RPS) that is a series of matches of the games. Each state of the system is expressed as follows:

`{(pair: n(`X`,`Y`)) (result: `Z`)}`

where X is your current choice, Y is the computer's current choice and Z is the result of the match, where each of X and Y is one of `rock`, `paper` and `scissors`, and Z is one of `win`, `lose` and `draw`. Let us suppose that X, Y and Z are initially set to `rock`, `rock` and `draw`. Given a state `{(pair: n(`X`,`Y`)) (result: `Z`)}`, each of X and Y is randomly chosen from `rock`, `paper` and `scissors`. Once X and Y are fixed, we know the result Z from them. This can decide all state transitions of the state machine formalizing the system.

2.2. Maude

Maude [4] is a rewriting logic-based specification/programming language supporting both equational and rewriting logic. Maude makes it possible to describe soups, observable components and braced soups of observable components. When M is specifying in Maude, T is specified as rewrite rules. A rewrite rule starts with the keyword `rl`, followed by a label enclosed with square bracket and a colon, two patterns (terms that may contain variables) connected with =>, and ends with a full stop. A conditional rule starts with the keyword `crl` and has a condition with the keyword `if` before full stop. The following are forms of a rewrite rule and conditional rewrite rule:

`rl` $[lb] : l$ `=>` r .

where lb is a label. An instance of l is replaced with the corresponding instance of r.

`crl` $[lb] : l$ `=>` r `if` $\ldots$ `/\` c_i `/\` $\ldots$.

where lb is a label and c_i is part of the condition, which may be an equation $lc_i = rc_i$ or a matching equation $lc_i := rc_i$. The negation of $lc_i = rc_i$ can be written as $(lc_i =/= rc_i) =$ `true`, where `= true` can be omitted. If the condition $\ldots$ `/\` c_i `/\` $\ldots$ holds, an instance of l is replaced with the corresponding instance of r.

The state transitions of RPS is specified as the following rewrite rule:

```
clr [game-match] : {(pair: n(X,Y)) (result: Z)}
=> {(pair: n(X1,Y1)) (result: Z1)}
if X1 Xs1 := rock paper scissors /\
   Y1 Ys1 := rock paper scissors /\
   Z1 := result(X1,Y1) .
```

where `rock paper scissors` is the associative-commutative collection of `rock`, `paper` and `scissors` and `result` is the function that judges the game match based on `X1` and `Y1`. The first two matching equations in the condition randomly choose one of `rock`, `paper` and `scissors` and assign it to each of `X1` and `Y1`, and the third matching equation uses the function `result` with `X1` and `Y1` and assigns the result to `Z1`.

Maude is equipped with model checking facilities (a reachability analyzer and an LTL model checker). Maude provides the search command that allows finding a state reachable from s such that the state matches pattern p and satisfies condition c:

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

where `MOD` is the name of the Maude module specifying the state machine under model checking, `n` and `m` are optional arguments stating a bound on the number of desired solutions and the maximum depth of the search, respectively. `n` typically is 1 and s is a given state (typically an initial state of the state machine). p is pattern and c is a condition. The condition part `such that` c can be omitted. The search command searches the reachable states from s for at most n states that can match the pattern p and make the condition c true. In this paper, Maude search command is used to confirm the characteristics guessed by observing graphical animations of NSLPK.

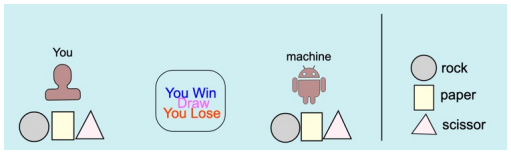


Figure 1: A state picture design for RPS

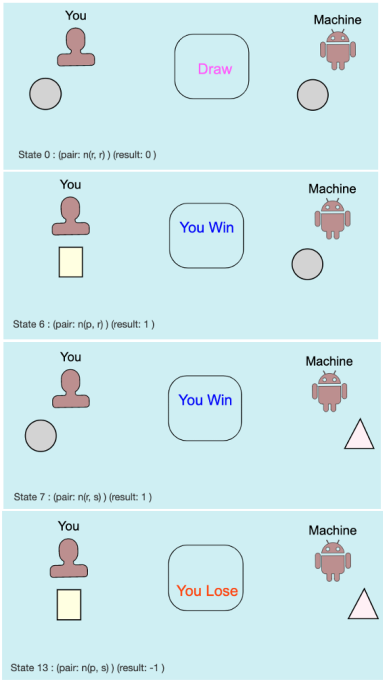


Figure 2: Four state pictures of a graphical animation for RPS

2.3. State machine graphical animation (SMGA)

State machine graphical animation tool (SMGA) is developed by Nguyen and Ogata [11]. The main purpose of SMGA is to help human users be able to recognize state patterns and perceive non-trivial characteristics of a state machine by observing its graphical animations. SMGA cannot automatically produce visual state picture designs and then it allows us to design a good state picture. As an input, SMGA basically takes a state picture design made by humans and a finite state sequence input generated by Maude. An output is a graphical animations by regarding the finite state sequence as a movie film based on the state picture design. One possible state picture design for the state machine formalizing RPS is shown in Figure 1. Figure 2 shows a four consecutive state pictures from the initial state.

3. NS(L)PK Protocols

NSPK [10] is a public-key authentication protocol designed by Needham and Schroeder and NSLPK [8] is a revised version of NSLPK by Low. NSLPK can be described as the following three message exchanges:

Init $\quad p \to q : \mathcal{E}_q(n_p, p)$

Resp $q \rightarrow p : \mathcal{E}_p(n_p, n_q, q)$
Ack $p \rightarrow q : \mathcal{E}_q(n_q)$

NSLPK uses public-key cryptography. Each principal, such as p and q, has a private/public key pair. The public key is known by all principals but the private one is only available to its owner. $\mathcal{E}_p(m)$ denotes the ciphertext obtained by encrypting a message m with the principal p's public key. n_p is a nonce (a random number) generated by principal p. A nonce is a unique and non-guessable number that is used only once in all protocol runs. The difference between NSLPK and NSPK is only the Resp message. NSPK uses $\mathcal{E}_p(n_p, n_q)$ as the Resp message.

If p wants to mutually authenticate q, p generates a nonce n_p and sends to q an Init message that consists of n_p and its ID p encrypted by the public key of q. When q receives the Init message, q tries to decrypt the ciphertext received by its private key. q then generates a nonce n_q and sends back to p a Resp message that consists of n_p, n_q, and ID q encrypted by the public key of p. On receipt of the Resp message, p tries to decrypt the ciphertext received by its private key. If the decryption is successful, p obtains two nonces and a principal ID and checks if the principal ID equals q and one of the nonces is the exact one that p has sent to q in this session. p then sends back to q an Ack message that contains n_q encrypted by the public key of q. On receipt of the message, q decrypts it, obtains a nonce and checks if the nonce is the one that q has sent to p in this session.

4. Formal Specification of NSLPK

We first introduce the following three operators (constructors) to represent three kinds of ciphertexts used in the protocol:

```
op enc1 : Prin Nonce Prin -> Cipher1 [ctor] .
op enc2 : Prin Nonce Nonce Prin -> Cipher2 [ctor] .
op enc3 : Prin Nonce -> Cipher3 [ctor] .
```

where `Prin` is the sort representing principals; `Nonce` is the sort denoting nonces; `Cipher1`, `Cipher2`, and `Cipher3` are the sorts denoting three kinds of ciphertexts contained in Init, Resp, and Ack messages, respectively. Given principals p, q and a nonce n_p, the term `enc1`(q, n_p, p) denotes the ciphertext $\mathcal{E}_q(n_p, p)$. `enc2` and `enc3` can be understood likewise. Hereinafter, let us use `Cipher1` (or `Cipher2`, or `Cipher3`) ciphertexts to refer to the ciphertexts contained in Init (or Resp, or Ack) messages.

The following operator (constructor) is used to represent nonces:

```
op n : Prin Prin Rand -> Nonce [ctor] .
```

where the third argument `Rand` is the sort denoting random numbers that makes the nonce globally unique and unguessable. Given principals p, q and a random value r, the term `n`(p, q, r) denotes a nonce created by p for q.

The following three operators (constructors) are used to represent the three kinds of messages used in NSLPK:

```
op m1 : Prin Prin Prin Cipher1 -> Msg [ctor] .
op m2 : Prin Prin Prin Cipher2 -> Msg [ctor] .
op m3 : Prin Prin Prin Cipher3 -> Msg [ctor] .
```

where `Msg` is the sort denoting messages. Given three principals c, s, r and a `Cipher1` ciphertext $ciph1$, the term `m1`$(c, s, r, ciph1)$ denotes an Init message such that c is the actual creator of the message, s is the seeming sender of the message, r is the receiver of the message and $ciph1$ is the message body. c may or may not be the same as s. If c is different from s, then the message must have been faked by the intruder. `m1` and `m2` can be understood likewise.

The network is modeled as associative-commutative collections of messages, which the intruder can use as his/her storage. Any message that has been sent or put once into the network is supposed to be never deleted from the network because the intruder can replay the message repeatedly, although the intruder cannot forge the first argument. Consequently, the empty network (i.e., the empty collection) means that no messages have been sent.

Let ms, rs, ns, and ps be collections of messages, random numbers, nonces, and principals, respectively. ps contains the intruder. Let $c1s$, $c2s$, and $c3s$ be collections of `Cipher1`, `Cipher2`, and `Cipher3` ciphertexts, respectively. To formalize the NSLPK protocol as a state machine M_{NSLPK}, we use the following observable components:

- (`nw` : ms) - it says that ms consists of all messages sent by principals and faked by the intruder;
- (`cenc1` : $c1s$) - it says that $c1s$ is the collection of the `Cipher1` ciphertexts gleaned by the intruder;
- (`cenc2` : $c2s$) - it says that $c2s$ is the collection of the `Cipher2` ciphertexts gleaned by the intruder;
- (`cenc3` : $c3s$) - it says that $c3s$ is the collection of then `Cipher3` ciphertexts gleaned by the intruder;
- (`nonces` : ns) - it says that ns is the the collection of nonces gleaned by the intruder;
- (`prins` : ps) - it says that ps is the collection of the principals participating in the protocol;
- (`rand` : rs) - it says that rs is the collection of random numbers available.

Each state in S_{NSLPK} is expressed as $\{obs\}$, where obs is a soup of those observable components. We suppose that three principals `p`, `q` and `intr` participate in the protocol, where `p` and `q` are trustable principals and `intr` is the intruder, and two random numbers `r1` and `r2` are initially available. Then, I_{NSLPK} consists of one initial state that is expressed as follows:

```
{(nw: emp) (rand: (r1 r2)) (nonces: emp)
 (cenc1: emp) (cenc2: emp) (cenc3: emp)
 (prins: (p q intr))} .
```

where `emp` denotes the empty collection.

Three rewrite rules `Challenge`, `Response`, and `Confirmation` formalize three actions when a principal sends an Init, a Resp, and an Ack message, respectively. Let `OCs` be a Maude variable of observable component soups; `P` & `Q` be Maude variables of principals; `Ps` be a Maude variable of collections of principals; `NW`, `R`, and `N` be Maude variables of collections of messages, random numbers and nonces, respectively; `Rs`, `CE1`, and `Ns` be Maude variables of collections of random numbers, `Cipher1` ciphertexts, and nonces, respectively. The rewrite rule `Challenge` is defined as follows:

```
rl [Challenge] : {(nw: NW) (prins: (P Q Ps))
(rand: (R Rs)) (cenc1: CE1) (nonces: Ns) OCs}
=> {(nw: (m1(P,P,Q,enc1(Q,n(P,Q,R),P)) NW))
(cenc1: (if Q == intr then CE1 else
(enc1(Q,n(P,Q,R),P) CE1) fi)) (nonces:
(if Q == intr then (n(P,Q,R) Ns) else Ns fi))
(rand: Rs) (prins: (P Q Ps)) OCs} .
```

The rewrite rule says that when `R` is in `rand`, a new Init message is put into the network, `R` is deleted from `rand`, the intruder gleans the nonce `n(P,Q,R)` if `Q` is the intruder and the intruder gleans the ciphertext `enc1(Q,n(P,Q,R),P)` if `Q` is not the intruder.

In addition to the three rewrite rules that formalize sending messages exactly following the protocol, we also introduce six more rewrite rules to formalize the intruder's faking messages:

- `fake12`, `fake22`, and `fake32`: a ciphertext `C` is available to the intruder, the intruder fakes and sends an Init, or a Resp, or an Ack message using `C`;
- `fake11` and `fake31`: a nonce `N` is available to the intruder, the intruder fakes and sends an Init or an Ack message using `N`;
- `fake21`: two nonces `N1` and `N2` are available to the intruder, the intruder fakes and sends a Resp message using `N1` and `N2`.

The rewrite rule `fake11` is defined as follows:

```
rl [fake11] : {(nw: NW) (nonces: (N Ns))
(prins: (P Q Ps)) OCs} =>
{(nw: (m1(intr,P,Q,enc1(Q,N,P)) NW))
(nonces: (N Ns)) (prins: (P Q Ps)) OCs} .
```

The rewrite rule says that when `N` is in `nonces`, a new intruder's faking Init message is put into the network.

5. State Picture Design of NSLPK Protocol

The network component, which consists of many messages, is the main part of the protocol that we should focus on. Initially, we try to make a design for the network in which Bui and Ogata [2] used, as shown in Figure 3. The

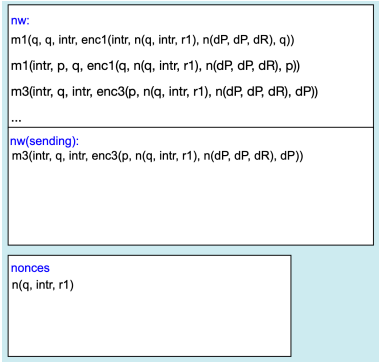


Figure 3: A simple state picture for the NSLPK protocol (1)

design, however, is hard to observe and/or analyze the messages in the network because there are many contents inside each message. As shown in Figure 3, there are three rectangles in which the first rectangle represents a network that contains all messages, the second one displays the most recent message that has been put into the network, and the collection of *nonces* gleaned by the intruder is displayed in the last rectangle. "..." is displayed whenever the content of the network is overflowed. During making a better state picture design, by observing that the number of messages increases by one after each state, we come up with an idea that displays the contents of the most recent message that has been put into the network (hereinafter, let us call such a message as the latest message).

Although there are three kinds of ciphertexts (i.e., `enc1`, `enc2`, and `enc3`), in the state picture design, we use only one form to visualize ciphertexts. The form is as follows: enc*i*(public-key, nonce1, nonce2, cipher-creator), where public-key is a principal (possibly `intr`), nonce1 for m1, m2, and m3 is in the following form: nonce1(generator, random, forwhom); nonce2 is in the following form: nonce2(generator, random, forwhom). When the ciphertext is in the form of `enc3`, cipher-creator receives a dummy principal `dP` as its value. Similarly, when the ciphertext is in the form of `enc1` or `enc3`, nonce2 receives a dummy value denoted by nonce2(`dP`,`dR`,`dP`), where `dR` denotes a dummy random number.

One possible way to observe&analyze the network is to observe&analyze each message in the network. Observing each message in the network is also equivalent to observe the latest message. Explicitly displaying the detailed content of the latest message helps us guess some non-trivial characteristics, which is discussed in Sect. 6. Furthermore, we design three sub-networks for three types of messages instead of one network that contains all messages. One network that contains all messages is another possible way to make the state picture design. Each way of design has some advantages as well as disadvantages. As shown in Figure 3, putting all messages in one place is simple but it is hard to distinguish each message. Designing three sub-networks for three types of messages helps us be able to immediately rec-

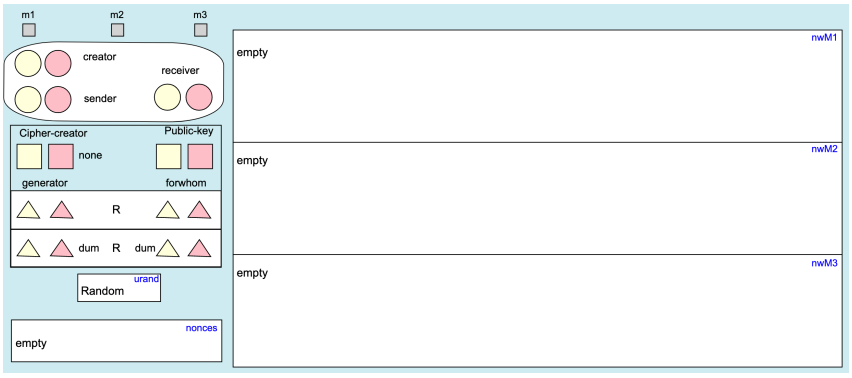


Figure 4: A state picture design for the NSLPK protocol (1)

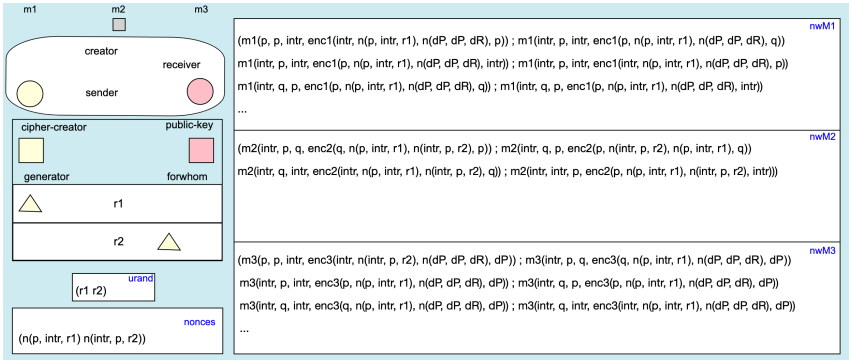


Figure 5: A state picture for the NSLPK protocol (1)

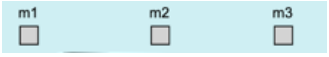
ognize the specific message type in each specific network. It makes us more transparent in our visual perception when we observe each specific message or the order/relation between messages.

In addition to the observable components presented in Sect. 4, some more observable components are introduced for visualizing the state picture design. They are as follows:

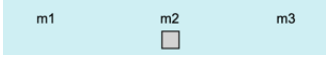
Observable components	Description
`newmsg`	The latest message (m1,m2,m3)
`m1`	Latest message `m1`
`m2`	Latest message `m2`
`m3`	Latest message `m3`
`nwM1`	Network contains messages `m1`
`nwM2`	Network contains messages `m2`
`nwM3`	Network contains messages `m3`
`urand`	Used random numbers
`nonces`	Nonces gleaned by intruder

Figure 4 depicts our state picture design. Some designs are used from state picture design tips of the work [3]. Figure 5 displays a state picture. We first divide two roles that are creators and senders into two separate places. Then, observable components are put to the corresponding place in which their roles seem to belong. For example, public-key should be put to the receiver's side because the sender uses the public-key of the receiver for encrypting. Values are displayed with different colors and shapes. For example, pink and light yellow colors represent two different principals, blank represents `intr`, triangles represent the contents of the nonce.

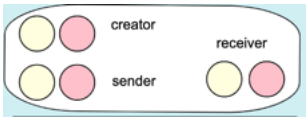
We describe the details of the state picture design. The representation of the three types of messages designed in Figure 4 is as follows:



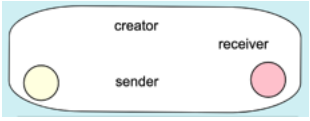
The type of the latest message is represented by a small light gray square. For example, when the latest message is a message `m2`, there is only one light gray square displayed under m2 as shown in the following picture:



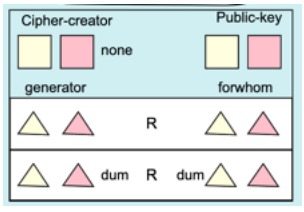
The representations of the creator, sender, and receiver of the message used in Figure 4 are as follows:



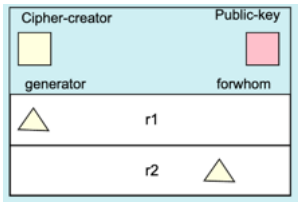
The creator of the message appears at the top-left place, pink and light yellow circles represent two different principals `q` and `p`. If the value is `intr`, nothing is displayed. The sender and receiver of the message appear at the bottom-left and bottom-right places, respectively. For example, when creator is `intr`, sender is `p`, receiver is `q`, it is displayed as follows:



The representations of the contents of the ciphertext shown in Figure 4 are as follows:



The cipher-creator of the ciphertext appears at the top-left place of the rectangle, pink and light yellow squares represent two principals `q` and `p`, respectively. If the value is `intr`, nothing is displayed. For the case the message is a message `m3`, the text "none" is displayed. The public-key of the ciphertext appears at the top-right place. If the value is `intr`, nothing is displayed. The two nonces of the ciphertext are shown with two rectangles inside the primary rectangle, where the upper rectangle visualizes the first nonce and the lower rectangle visualizes the second nonce. In the first nonce, the generator and forwhom representations appear at the left-hand side and right-hand side, respectively; pink and light yellow triangles are the principals `q` and `p`, respectively. If the value is `intr`, nothing is displayed. The random representation appears at the middle place in which the random number value used is displayed. The second nonce is represented likewise. If the message is a message `m3`, the text `dum` is displayed for the values of generator and forwhom, where `dum` denotes the dummy value `dP`. Considering the following example. cipher-creator is `p` and public-key is `q`. In the first nonce generator is `p`, random is `r1`, and forwhom is `intr`. In the second nonce, generator is `intr`, random is `r2`, and forwhom is `p`. Those values are displayed as follows:



In Figure 4, the representations of urand and nonces are designed at the left-bottom corner. The values of both urand and nonces are displayed using two rectangles as follows:

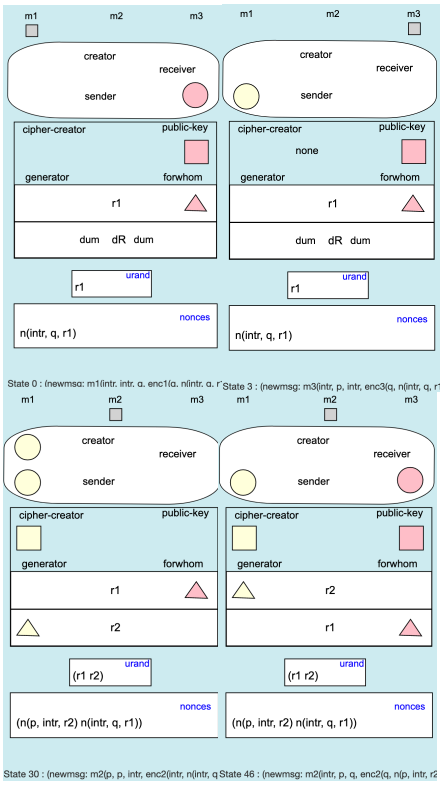
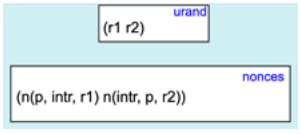
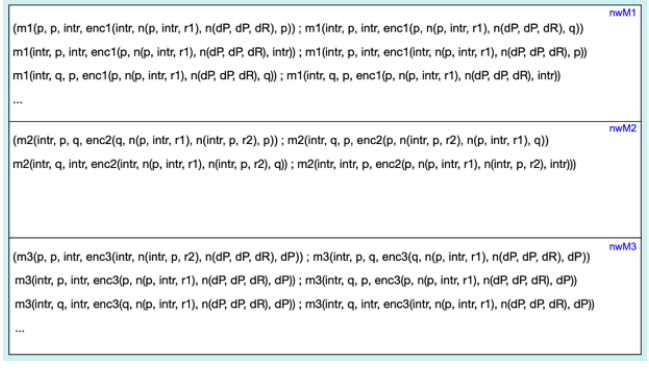


Figure 6: Some state pictures for the NSLPK protocol (1)



In Figure 4, three types of network representations are designed on the right side. "…" is displayed whenever the messages are overflowed. This can be seen in the figure below:



6. Characteristics Guessed Based on Our Design

This section presents how to guess the characteristics of NSLPK by observing graphical animations using SMGA. Observing graphical animations of a state machine allows

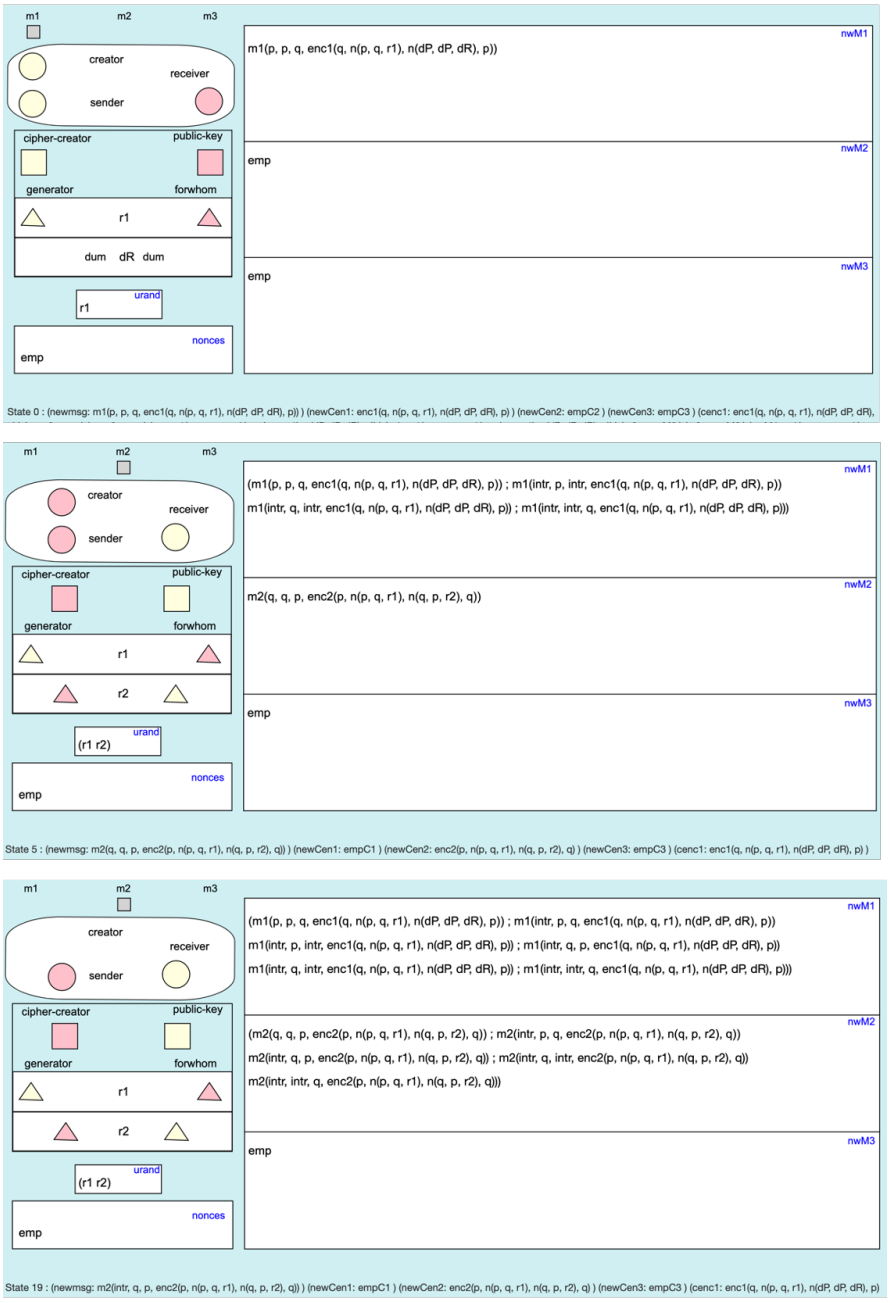


Figure 7: Some state pictures for the NSLPK protocol (2)

users to recognize some relations between observable components (OCs). Observing all OCs at the same time is less likely to recognize the characteristics since there are many OCs in the design picture.

There are some tips on how to conjecture characteristics of NSLPK by observing graphical animations with SMGA as follows:

1. By concentrating on one observable component, we may find that if the value of that observable component is `intr`, any other observable components may have some specific values, from which we may conjecture some characteristics.
2. By concentrating on two different observable components, we may find a relation between them, from which we may conjecture some characteristics.
3. By observing the order of the message in the network, we may find a relation between them, from which we may conjecture some characteristics.
4. By carefully investigating the conditions of some characteristics that have been already conjectured, we may conjecture some other characteristics.

Hence, we sometimes need to concentrate on some specific OCs when we observe the graphical animations. Characteristics of NSLPK that involve one message are straightfor-

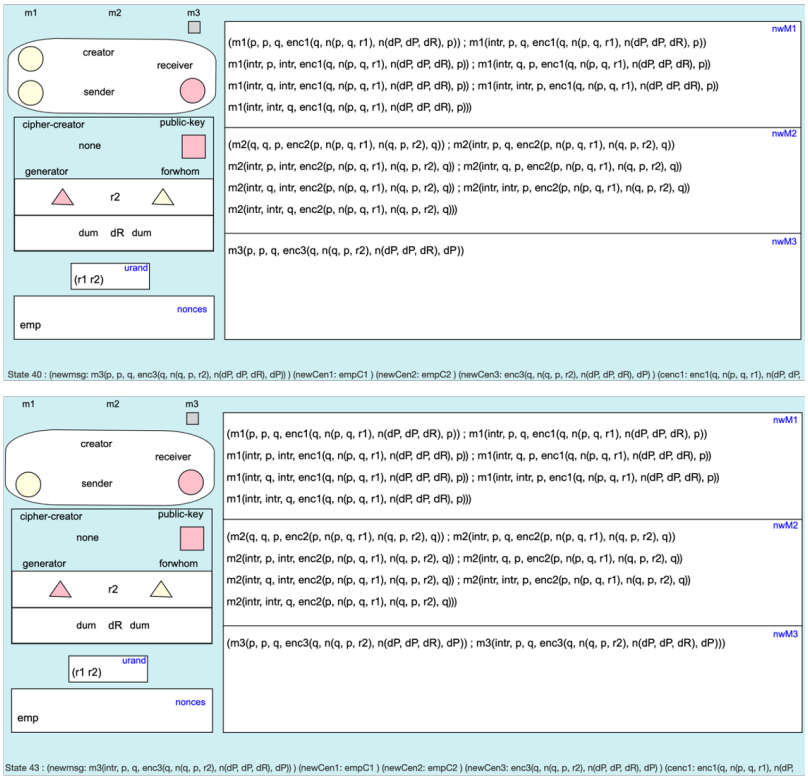


Figure 8: Some state pictures for the NSLPK protocol (3)

ward to guess by observing graphical animations. However, characteristics of NSLPK that involve two or more messages some are not.

Figure 6 shows four pictures of states for M_{NSLPK}. Taking a look at the first picture (of State 0) and the second picture (of State 3) helps us recognize that there is `n(intr,q,r1)` in `nonces` when `generator` is `intr` and taking a look at the third picture (of State 30) and the fourth picture (of State 46) helps us recognize that there is `n(p,intr,r2)` in `nonces` when `forwhom` is `intr`. Any nonce gleaned by the intruder is stored in `nonces`. Hence, observing the graphical animation of these four pictures helps us guess the characteristic such that any nonce gleaned by the intruder has been generated by the intruder or a non-intruder principal that wanted to authenticate the intruder.

Taking a look at the second picture (of State 3) and the third picture (of State 30) allows us to guess another characteristic such that whenever `receiver` is `intr` (that displays blank in the state pictures) in the latest message, then the nonce of that message is in `nonces`. Carefully observing graphical animations helps us perceive one more characteristic. Taking a look at the four pictures of Figure 6, we recognize the characteristic that when a nonce is in `nonces`, the random number used in the nonce is stored in the collection of used random numbers `urand`.

We prepare another input file that consists of a finite sequence of states so that we can guess more characteristics by observing the behavior of the protocol. To guess some non-trivial characteristics, we observe the information in which the order of the messages is mainly focused on. Carefully observing graphical animations of the order of messages in the network and expecting that a message `m2` should follow a message `m1`, we guess a characteristic which includes two messages as shown in Figure 7. Taking a look at the first picture (of State 0), there exists a message `m1(p,p,q,enc1(q,n(p,q,r1),n(dP,dP,dR),p))` in `nwM1`. After some `m1` messages are faked by the intruder based on the gleaned information, there exists a message `m2(q,q,p,enc2(p,n(p,q,r1), n(q,p,r2),q))` in `nwM2` at the second picture (of State 5). Taking a look at the third picture (of State 19), we observe that the intruder creates many faked `m2` messages including `m2(intr,q,p,enc2(p,n(p,q,r1),n(q,p,r2), q))`. Observing the order of messages in the network allows us to conjecture the following characteristic:

> If there exists a message `m1` created by a non-intruder principal and sent to another non-intruder principal, and
>
> there exists a message `m2` (either created by the intruder or a non-intruder principal) that is sent to the sender of `m1`,
>
> then the message `m2` originates from a non-intruder principal who is the receiver of the `m1`.

Similarly, we expect that a message `m3` should follow a message `m2`. There is a message `m2(q,q,p, enc2(p,n(p,q,r1),n(q,p,r2),q))` in `nwM2` at the second pic-

ture (of State 5). Taking a look at the first picture (of State 40) in Figure 8, there exists a message `m3(p,p,q,enc3(q,n(q,p,r2),n(dP,dP,dR), dP))` in `nwM3`. At the second picture (of State 43), there exists a message `m3(intr,p,q, enc3(q,n(q,p,r2),n(dP,dP,dR),dP))` in `nwM3` which is created by `intr`. Carefully observing the order of the messages in the network, we also guess the following characteristic:

If there exists a message m2 created by a non-intruder principal and sent to another non-intruder principal, and there exists a message m3 (either created by the intruder or a non-intruder principal) that is sent to the sender of the message m2,
then the message m3 originates from the non-intruder principal who is the receiver of the message m2.

Observing graphical animations of the NSLPK produced by SMGA could help us visually perceive several characteristics. The informal descriptions of the guessed characteristics are as follows:

1. If the latest message is a message `m1` and `cipher-creator` of `m1` is `intr`, then the nonce of `m1` is in `nonces` (i.e., the nonce is gleaned by the intruder).
2. If the latest message is a message `m1` that forms as `m1(p,p,q,enc1(q,n,p))` and `p` is not `intr`, then the forwhom of `n` is `q`.
3. If the latest message is a message `m2` that forms as `m2(p,p,q,enc2(q,n1,n2,p))` and `p` is not `intr`, then the forwhom of `n2` is `q`.
4. If the latest message is a message `m3` that forms as `m3(p,p,q,enc3(q,n))`, and `p` and `q` are not `intr`, then the generator of `n` is `q`.
5. If `public-key` of the latest message is `intr`, then a nonce/nonces in that message is/are in `nonces`.
6. If a nonce is in `nonces`, then either generator or forwhom of the nonce is `intr`.
   - If generator of a nonce is `intr`, the nonce is in `nonces`.
   - If generator and forwhom of a nonce are not `intr`, then the nonce is not in `nonces`.
7. If a nonce in the latest message forms as `n(p,q,r)`, and `p` is not `intr`, then `r` is in urand.
8. if a nonce is in `nonces`, then random of the nonce is in urand.
9. If message `m1(p,p,q,enc1(q,n(p,q,r),p))` is in `nwM1` and message `m2(q1,q,p, enc2(p,n(p,q,r),n,q))` is in `nwM2` and `p` is not `intr` then `m2(q,q,p,enc2(p,n(p,q,r), n,q))` is in the network and originates from `q`.
10. If message `m2(q,q,p,enc2(p,n,n(q,p,r),q))` is in `nwM2` and message `m3(p1,p,q, enc3(q, n(q,p,r))` is in `nwM3`, and `q` is not `intr` then `m3(p,p,q,enc3(q,n(q,p,r))` is in the network and originates from `p`.

Maude search command can be used as an invariant model checker to check that the NSLPK protocol enjoys the guessed characteristics. The guessed characteristics are confirmed by the search command at a specific depth (depth 5) of the state space because the reachable state space (generated by Maude) of the protocol is too huge to be exhaustively traversed. The search command does not find any counterexample at depth 5. It means that the NSLPK protocol seems to enjoy the guessed characteristics.

7. Graphical Animation in Sequence Diagram

7.1. Idea

Sequence diagram is used to model the interactive behavior system entities, which is one of the most used diagrams of UML [13]. Besides, message sequence charts (MSCs) are widely used to capture system requirements during the early design stages [6]. A variant of MSCs is also called sequence diagrams used in UML. The Alice-Bob format that is often used to describe security protocols is a kind of sequence diagrams. Therefore, sequence diagram is one possible way to visualize message exchanges between principals in authentication protocols. We develop SMGA-SD that automatically generate a sequence diagram from a sequence of states and integrate SMGA-SD with SMGA. Regarding security protocols, such as NSLPK, messages from a principal are not delivered immediately to the recipient but stored in the network so that intruders can intercept and/or replay messages. Hence, our sequence diagrams are designed slightly differently from standard sequence diagrams in that a message is not delivered immediately to the recipient. We suppose that messages in the network never be deleted. The behavior of intruders makes protocols unpredictable, which may lead to the middle-person attack [8]. To express principals (including the intruder) in NSPK and NSLPK, we draw three parallel vertical lines denoting three principals where one principal is intruder whose line is in the middle, and two others represent two trustable principals. Although we can draw as many principals as many vertical lines in SMGA-SD, for simplicity, we keep the current appearance of our sequence diagrams. Horizontal arrows represent messages exchanges between principals. The message content is displayed above in the middle position of the arrow. There are two kinds of messages in which (1) one follows the protocol and (2) the other is faked by the intruder. To distinguish the two kinds of messages, the blue color is used for (1), while the red color is used for (2). Some functionalities for animations in SMGA are applied to SMGA-SD, such as *Run*, *Stop*, *Run step*, *Back step*, so that users can control the animation of the sequence diagram.

Let us describe two main different points of SMGA-SD compared to the standard sequence diagram as follows:

1. In standard sequence diagrams, the target of arrow messages is drawn directly to principals. In our diagram, we assume that messages sent are first put into the network. Those messages are intercepted and/or replayed by intruders mentioned above. Therefore, the target of arrow messages is not drawn directly to principals except for the case in which the recipient is in-

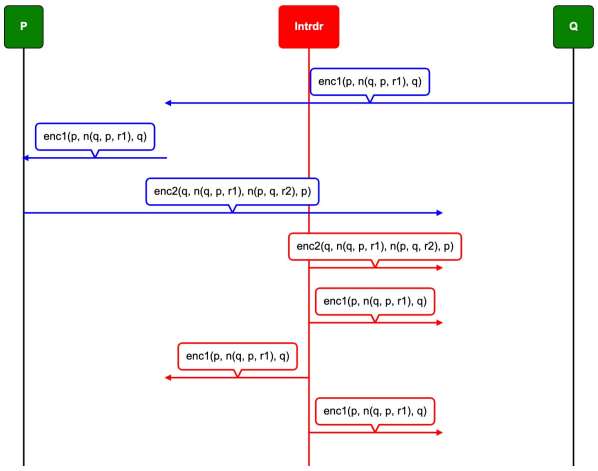


Figure 9: A snapshot of sequence diagram for NSLPK protocol

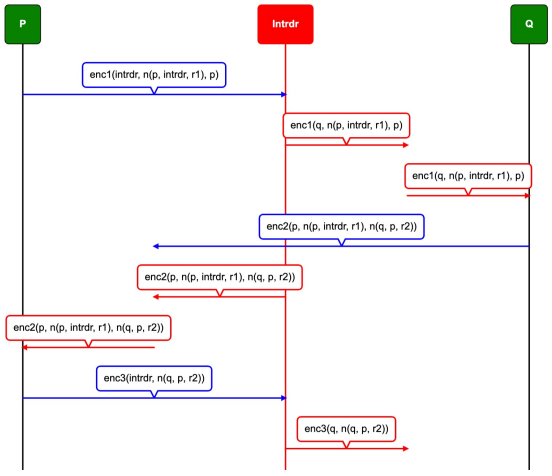
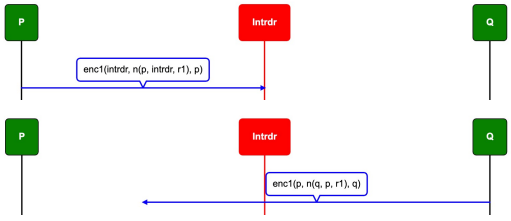


Figure 10: A sequence diagram for NSPK protocol

truder. Because the intruder can intercept all messages in the network, messages sending to the intruder can be regarded as received by the intruder without delay. The following figure shows two cases: principal `P` sends a message to intruder `Intrdr`, where the source of arrow message starts from `P` and ends at `Intrdr` at the top of the figure, and principal `Q` sends a message to principal `P` where the source of arrow message starts from `Q` and ends at the point between `P` and `Intrdr` at the bottom of the figure, which means that the message is not delivered to principal `P` yet, but it is stored in the network.



2. When a principal obtains a message (called a received message) from the network, depending on what the message is, the principal can produce a new message to reply to it. The figure below shows a case when principal `Q` sends a message to principal `P` (the first arrow) but has not delivered to `P` yet until principal `P` receives the message and produces a new message to send back to principal `Q` (in the second and third arrows, respectively). Of course, the message sent by principal `P` is not delivered to principal `Q` yet.

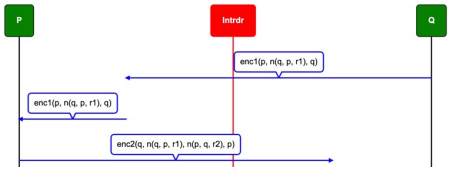
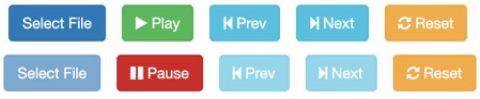


Figure 9 shows a sequence diagram of NSLPK. There are five buttons: *Select File*, *Play*, *Prev*, *Next*, *Reset* that correspond to five functionalities, which are the same as in SMGA, as follows:

- *Select File* to import a state sequence file.
- *Play* to draw a sequence diagram step by step with a speed selected by users.
- *Pause* to stop drawing a sequence diagram. When a user clicks *Play* button, the button becomes *Pause* button. The following figure shows the moment before and after clicking *Play* button.



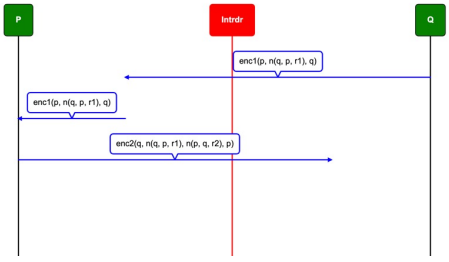
- *Next* to go forward to the next diagram in one step (one state transition).
- *Prev* to go back to the previous diagram in one step.
- *Reset* to reset the diagram to the beginning when a user just imports a state sequence.

7.2. Graphical Animations of NSLPK in Sequence Diagram

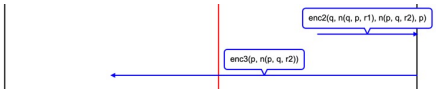
In the sub-section, we describe how to visualize NSLPK as sequence diagrams by SMGA-SD. The work flow is the same as SMGA. Firstly, we use Maude to formalize NSLPK and generate a state sequence as an input file. The input file is then imported into SMGA-SD. Apart from some existing observable components used in SMGA, we use a new observable component called `recmsg1` in the specification that represents the received message. All observable components used SMGA-SD are summarized as follows:

- (prins: (p q intrdr)): we use three values of prins to display the name of three principals on the diagram.
- (newmsg: message): the message format is same as what we defined in Sec. 4.
- (recmsg1: message): this observable component is used with newmsg to display the received message. This observable component stores the message that is m1 (in case the rl [Response] is used), m2 (in case the rl [Confirmation] is used), and empt (for the rest cases). Note that the intruder always gets the messages in the network as our assumption so we do not need to display the received message of the intruder. If recmsg1 observable component is m1, the newmsg observable component must be m2.

  An example below shows the newmsg and recmsg1 components in which newmsg is m2 and recmsg1 is m1:

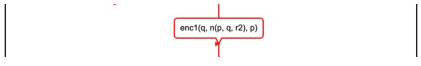


  If recmsg1 observable component is m2, the newmsg observable component must be m3. An example below shows the newmsg and recmsg1 observable components in which newmsg is m3 and recmsg1 is m2:



To be able to distinguish the messages following the protocol and those being faked by the intruder, we modify the content of messages in the specification in which a boolean value is added denoting that a message is faked by intruders or not. When drawing a message, we check the boolean value to decide the color of the message. If it is true, the red color is used. Otherwise, the blue color is used.

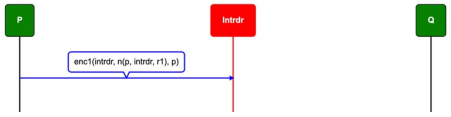
While observing the diagram for the first time, we have found that some states are meaningless, such as the intruder sends messages to himself/herself. Then, we modify the specification to avoid that situation. The following figure displays a message in which the intruder sends the message to himself/herself:



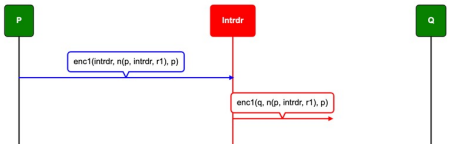
Looking at the diagram in Figure 9 makes us immediately distinguish two kinds of messages: one is message sent following the protocol and the other is those being faked by the intruder. We can recognize the sender and receiver of each message and comprehend their order.

7.3. Graphical Animations of NSPK in Sequence Diagram

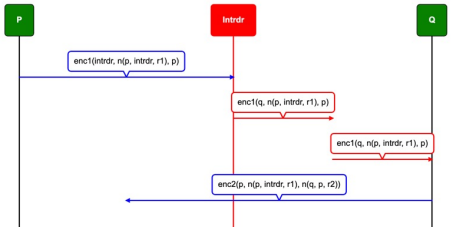
NSPK is flawed as detected by Lowe [8] and does not enjoy the nonce secrecy property (NSP). We use the Maude search command to find out a state sequence in which NSP is broken. The state sequence is then used to visualize a sequence diagram by SMGA-SD as in Figure 10. Looking at the diagram in Figure 10 makes it easier for us to understand why the property is broken. Note that by running animations step by step, we may better comprehend this flaw. Let us describe each state when using animations. Firstly, principal P sends message m1 to intruder as follows:



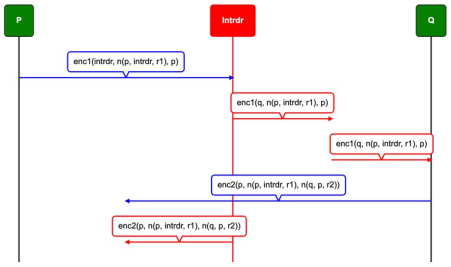
Then intruder fakes message m1 and sends it to principal Q as follows:



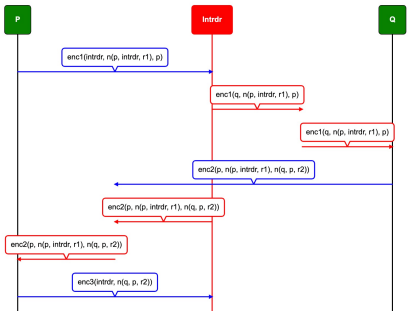
Whenever principal Q receives message m1 faked by intruder and the seeming sender is P, principal Q then sends message m2 to principal P as follows:



At this time, intruder can intercept message m2 and replay it to principal P as follows:



Whenever principal P receives message m3 faked by intruder and the seeming sender is intruder, it sends message m3 back to intruder as follows:



Finally, intruder fakes `m3` message and sends it to principal Q as shown in Figure 10.

8. Conclusion

We have graphically animated NSLPK and NSPK with SMGA and SMGA-SD. NSLPK has been visualized in a way that is different from the Alice-Bob format, while NSPK, together with NSLPK, has been basically visualized in the Alice-Bob format, although there are some differences between our sequence diagrams and standard sequence diagrams. Observing the graphical animations based on our first original design allows us to guess some (non-trivial) characteristics of the state machine formalizing NSLPK. We have checked the characteristics by Maude search command. Using the middle-person attack to NSPK, we have described how to use our sequence diagrams to help users comprehend why the attack is doable for NSPK. One piece of our future work is to graphically animate state machines that formalize other authentication protocols with SMGA, such as TLS [5].

Acknowledgment

The authors would like to thank the anonymous reviewers who carefully read an earlier version of the paper and gave them valuable comments without which they were not able to complete the present paper.

References

[1] Aung, M.T., Nguyen, T.T.T., Ogata, K., 2018. Guessing, model checking and theorem proving of state machine properties – a case study on Qlock. IJSECS 4, 1–18. doi:10.15282/ijsecs.4.2.2018.1.0045.

[2] Bui, D.D., Ogata, K., 2019. Graphical animations of the Suzuki-Kasami distributed mutual exclusion protocol. JVLC 2019, 105–115. doi:10.1007/978-3-319-90104-6_1.

[3] Bui, D.D., Ogata, K., 2020. Better state pictures facilitating state machine characteristic conjecture, in: DMSVIVA 2020, pp. 7–12. doi:10.18293/DMSVIVA20-007.

[4] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C. (Eds.), 2007. All About Maude. volume 4350 of *LNCS*. Springer. doi:10.1007/978-3-540-71999-1.

[5] Dierks, T., Allen, C., 1999. The TLS protocol version 1.0. RFC 2246, 1–80. doi:10.17487/RFC2246.

[6] Harel, D., Thiagarajan, P.S., 2003. Message Sequence Charts. Springer US, Boston, MA. pp. 77–105. URL: https://doi.org/10.1007/0-306-48738-1_4, doi:10.1007/0-306-48738-1_4.

[7] K. W. Brodlie, et al. (Ed.), 1992. Scientific Visualization: Techniques and Applications. Springer. doi:10.1007/978-3-642-76942-9.

[8] Lowe, G., 1995. An Attack on the Needham-Schroeder Public-Key Authentication Protocol. Inf. Process. Lett. 56, 131–133. doi:10.1016/0020-0190(95)00144-2.

[9] Mon, T.W., Bui, D.D., Duong, T.D., Ogata, K., 2021. Graphical animations of NSLPK authentication protocol, in: 27th DMSVIVA, pp. 39–45. doi:10.18293/DMSVIVA2021-005.

[10] Needham, R.M., Schroeder, M.D., 1978. Using Encryption for Authentication in Large Networks of Computers. Commun. ACM 21, 993–999. doi:10.1145/359657.359659.

[11] Nguyen, T.T.T., Ogata, K., 2017a. Graphical animations of state machines, in: 15th DASC, pp. 604–611. doi:10.1109/DASC-PICom-DataCom-CyberSciTec.2017.107.

[12] Nguyen, T.T.T., Ogata, K., 2017b. Graphically perceiving characteristics of the MCS lock and model checking them, in: 7th SOFL+MSVL, pp. 3–23. doi:10.1007/978-3-319-90104-6_1.

[13] Van Amstel, M.F., Lange, C.F., Chaudron, M.R., 2007. Four automated approaches to analyze the quality of uml sequence diagrams, in: 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), pp. 415–424. doi:10.1109/COMPSAC.2007.119.