Graphical Animations of the NSLPK Authentication Protocol

Thet Wai Mon, Dang Duy Bui, Duong Dinh Tran, Kazuhiro Ogata
School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)
1-1 Asahidai, Nomi, Ishikawa 923-1292, Japan
Email: {thetwaimon,bddang,duongtd,ogata}@jaist.ac.jp

***Abstract*—The behavior of the NSLPK authentication protocol is visualized using SMGA so that human users can visually perceive non-trivial characteristics of the protocol by observing graphical animations. These characteristics could be used as lemmas to formally verify that the protocol enjoys desired properties. We first carefully make a state picture design for the NSLPK protocol 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 correctness of the guessed characteristics using model checking. The work demonstrates that SMGA can be applied to the wider class of systems/protocols, authentication protocols in particular.**

***Keywords*-graphical animation; SMGA; NSLPK protocol; state machine; state picture design**

I. Introduction

SMGA [1] has been developed to visualize graphical animations of protocols. The main purpose of SMGA is to help human users be able to visually perceive non-trivial characteristics of the protocols by observing its graphical animations because humans are good at visual perception [2]. Those characteristics can be used as lemmas to formally prove that systems/protocols enjoy desired properties. Several case studies have been conducted on some protocols with SMGA. Among the protocols are shared-memory mutual exclusion protocols [3], [4], [5], a distributed mutual exclusion protocol [6], and a communication protocol [1]. Any authentication protocols have not been yet tackled with SMGA. It is worth tackling authentication protocols with SMGA because such protocols, such as TLS, are infrastructure in our highly networked environment.

We aim at coming up with a brand-new way to visualize the behavior of an authentication protocol called NSLPK [7]. Since it is known that state picture designs affect how well human users can detect non-trivial characteristics of protocols [5], we carefully make a state picture design of the NSLPK protocol and produce graphical animations of NSLPK based on the state picture design. By observing the graphical animations, some non-trivial characteristics are guessed by human users and checked with Maude [8]. In the paper, we mainly focus on how to make the state picture design of the NSLPK protocol and how some characteristics could be found by observing graphical animations with detailed experiments.

Bui and Ogata [6] have revised SMGA so as to visualize the network components in a distributed mutual exclusion protocol, which is applied partly to our work. VA4JVM [9] is a tool that can visualize outputs generated by Java Pathfinder (JPF). JPF outputs are often long and hard to read, especially when JPF finds something wrong, such as race-condition and deadlock. VA4JVM supports some functionalities, such as zooming, filtering, highlighting some specific parts of JPF outputs. Those functionalities can help human users observe some fragments that look interesting to be able to better comprehend JPF outputs. Counterexample generated by Maude LTL model checker can be graphically animated by SMGA [10]. Although Maude LTL model checker is a classical model checker and JPF is a software model checker, it would be worth considering some VA4JVM functionalities, such as zooming, filtering, and highlighting, to apply them to the future version of SMGA.

We assume that readers are familiar with state machines and Maude to some extent. NSLPK [7] is a modification of the NSPK authentication protocol [11]. The NSLPK protocol can be described as the following three message exchanges:

Init $\quad p \rightarrow q : \varepsilon_q(n_p, p)$
Resp $\quad q \rightarrow p : \varepsilon_p(n_p, n_q, q)$
Ack $\quad p \rightarrow q : \varepsilon_q(n_q)$

Each principal such as p and q has a private/public key pair, and the public counterpart is shared with all principals but the private one is only available to its owner. $\varepsilon_p(m)$ denotes the ciphertext obtained by encrypting the 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.

II. Formal Specification of NSLPK

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

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

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).

DOI reference number: 10.18293/DMSVIVA2021-005

where Prin is the sort representing principals; Nonce is the sort denoting the nonce numbers; 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 term $\texttt{enc1}(q, n_p, p)$ denote the ciphertext $\varepsilon_q(n_p, p)$ obtained by encrypting n_p and p with the principal q's public key. 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. A Nonce is defined by the following operator:

```
op n : Prin Prin Rand -> Nonce .
```

where the third argument Rand is the sort denoting random numbers that makes the nonce globally unique and unguessable. Given principals p, q and random value r, term $\texttt{n}(p, q, r)$ denote a nonce created by principal p for authenticating p to principal q.

We specify three kinds of messages used in the NSLPK protocol as follows:

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

where Msg is the sort denoting messages. m1, m2, and m3 are the sorts denoting three kinds of messages Init, Resp, and Ack, respectively. The first, second, and third arguments of each of m1, m2, and m3 are the actual creator, the seeming sender, and the receiver of the corresponding message. The first argument is meta-information that is only available to the outside observer and the principal that has sent the corresponding message, and that cannot be forged by the intruder; while the remaining arguments may be forged by the intruder.

The network is modeled as a multiset 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 can not forge the first argument. Consequently, the empty network (i.e., the empty multiset) means that no messages have been sent.

In this paper, a state is expressed as a soup of observable components. Let ms, rs, ns, and ps be the collections of messages, random numbers, nonces, and principals, respectively. ps may contain an intruder. Let $c1s$, $c2s$, and $c3s$ be the 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 the network is constructed by ms,
- (cenc1 : $c1s$) - it says that the collection of Cipher1 ciphertexts gleaned by the intruder is $c1s$,
- (cenc2 : $c2s$) - it says that the collection of Cipher2 ciphertexts gleaned by the intruder is $c2s$,
- (cenc3 : $c3s$) - it says that the collection of Cipher3 ciphertexts gleaned by the intruder is $c3s$,
- (nonces : ns) - it says that the collection of $nonces$ gleaned by the intruder is ns,
- (prins : ps) - it says that the principals participating in the protocol are ps,
- (rand : rs) - it says that the available random numbers are rs. Every time a principal wants to send an Init or a Resp message, it needs to generate a random and globally unique number. To formalize that behavior, we provide a fixed collection of random numbers from the beginning, and every time a process needs to generate a random number, an element is extracted and used.

Each state in S_{NSLPK} is expressed as $\{obs\}$, where obs is a soup of those observable components. We suppose that two principals p & q together with an intruder participate in the NSLPK protocol, one initial state of I_{NSLPK} namely init is defined as follows:

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

where intr is a constant of Prin denoting the intruder, and emp denotes an 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 denoting a network, a random number, and a nonce, respectively; Rs, CE1, and Ns be Maude variables denoting a collection of random numbers, Cipher1, 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, intruder gleans the nonce and the ciphertext used in that message if that message sends to the intruder, and R is removed from rand.

In addition to the three rewrite rules that formalize sending messages exactly following the protocol mentioned above, 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

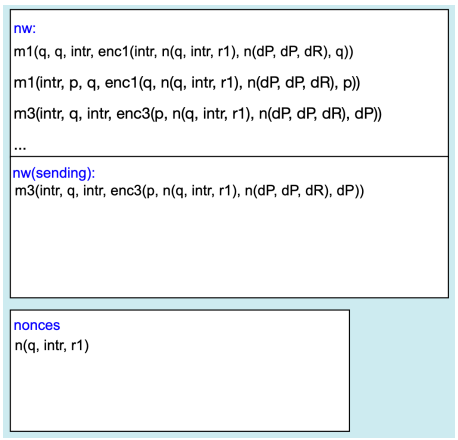


Figure 1. A simple state picture for the NSLPK protocol (1)

Init, or a Resp, or an Ack message using C, respectively.

- fake11 and fake31: a nonce N is available to the intruder, the intruder fakes and sends an Init or an Ack message using N, respectively,
- 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)) (cenc1: CE1) OCs} =>
{(nw: (m1(intr,P,Q,enc1(Q,N,P)) NW)) (cenc1:
(if Q == intr then CE1 else (enc1(Q,N,P) CE1)
fi)) (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, and the intruder gleans the ciphertext sent in that message.

The remaining rewrite rules can be defined likewise.

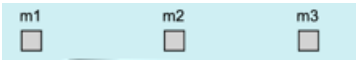
III. State Picture Design of NSLPK Protocol

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 [6] used, as shown in Fig. 1. The 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 Fig. 1, 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.

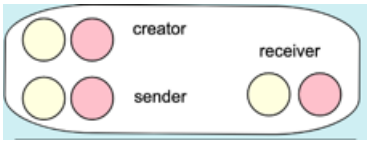
Fig. 2 depicts our state picture design. Some designs are used from state picture design tips of the work [5]. Fig. 3 displays a state picture. We first divide two roles that are creators and senders into two separate places. Then, observable components are put to the corresponding place in which their 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 Fig.2 is as follows:



The type of the lastest message is represented by a small light gray square. For example, when the lastest 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 Fig. 2 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:

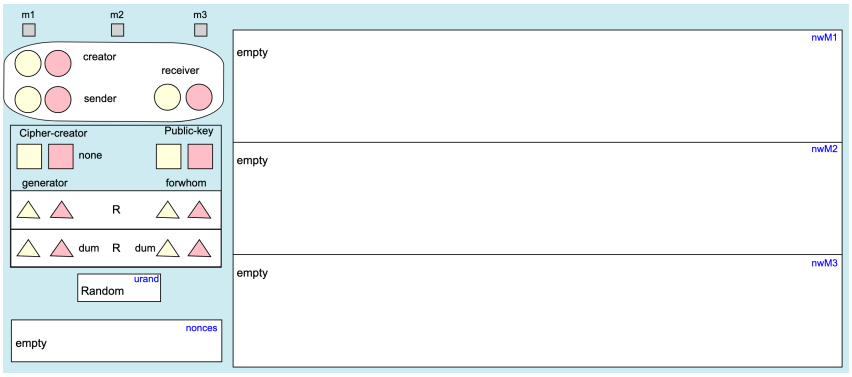


Figure 2. A state picture design for the NSLPK protocol (1)

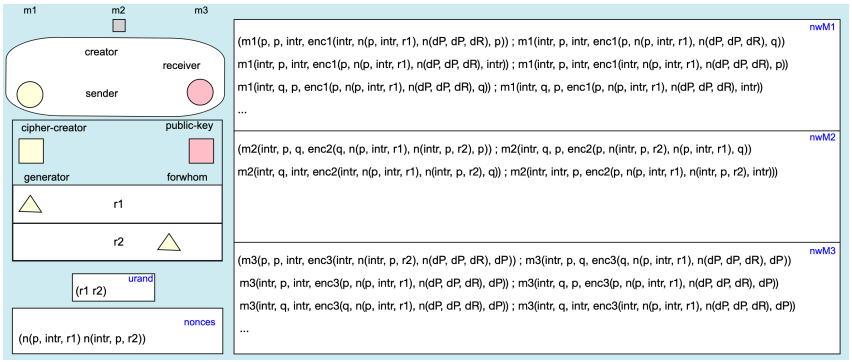
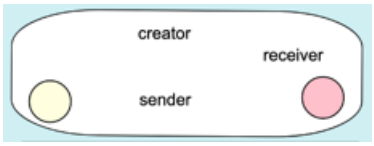
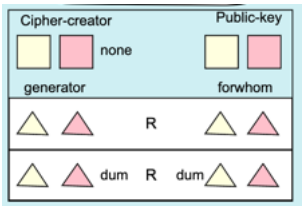


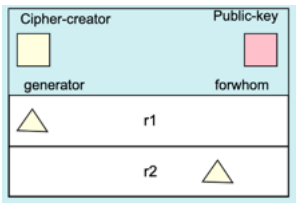
Figure 3. A state picture for the NSLPK protocol (1)



The representations of the contents of the ciphertext shown in Fig. 2 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 Fig. 2, 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:

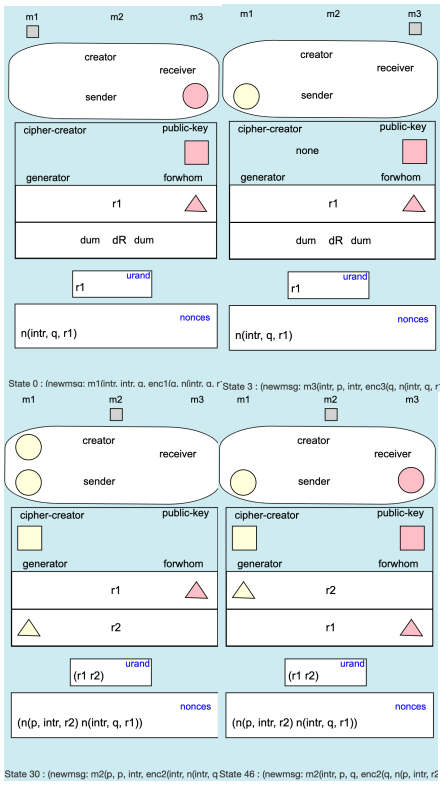
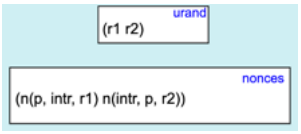
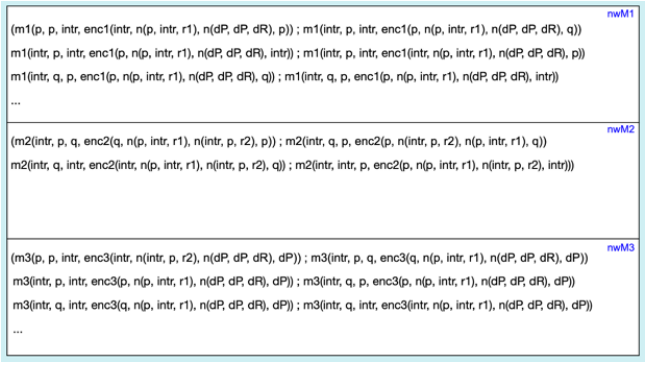


Figure 4. Some state pictures for the NSLPK protocol (1)



In Fig. 2, 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:



IV. Characteristics Guessed Based on Our Design

We sometimes need to concentrate on some specific OCs when we observe the graphical animations. Most of the characteristics of the NSLPK protocol are straightforward to guess by observing graphical animations. However, some are not, precisely the characteristics that include two messages. We use Maude to generate a finite input sequence of states based on the Maude specification of the protocol, then feed it to SMGA which produces graphical animation of the input sequence of states.

Fig. 4 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 Fig. 4, 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 concentrate on the order in which messages have been sent. Carefully observing the order of messages, especially that a message `m2` should follow a message `m1`, as Fig. 5, we guess a characteristics that involves two messages. 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`.

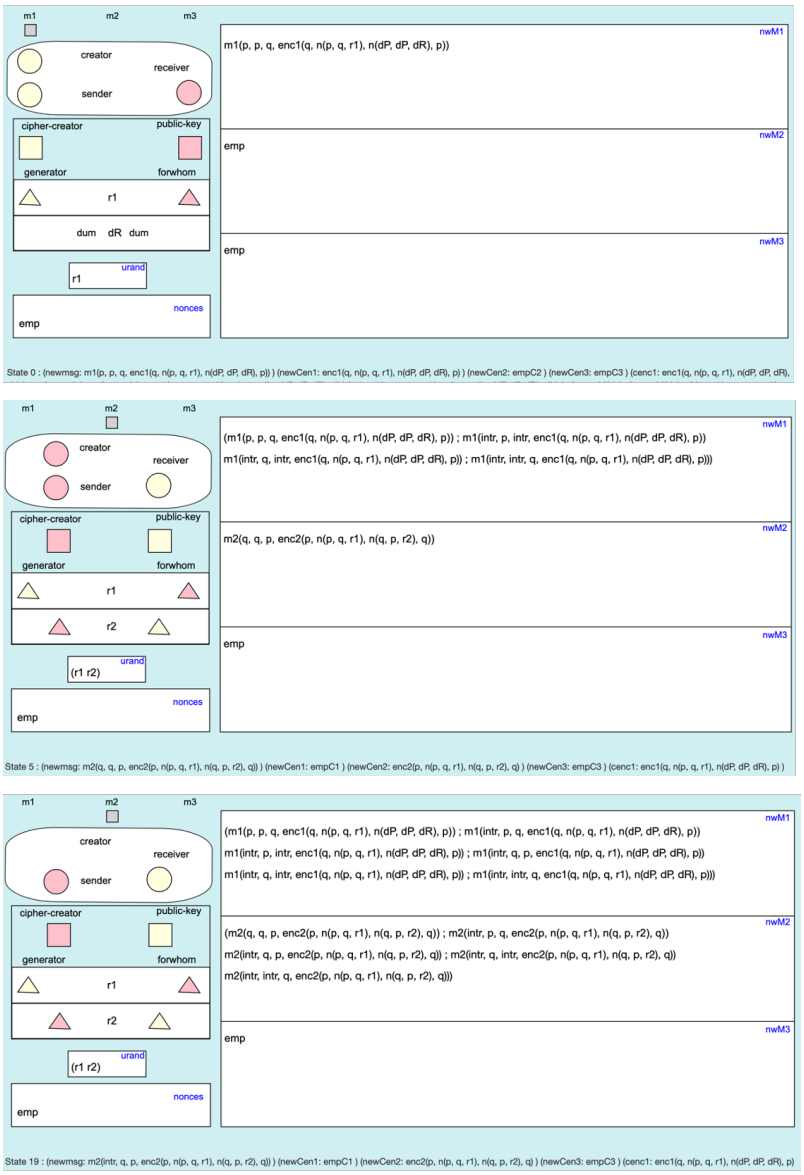


Figure 5. Some state pictures for the NSLPK protocol (2)

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 picture (of State 5). Taking a look at the first picture (of State 40) in Fig. 6, 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`.

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

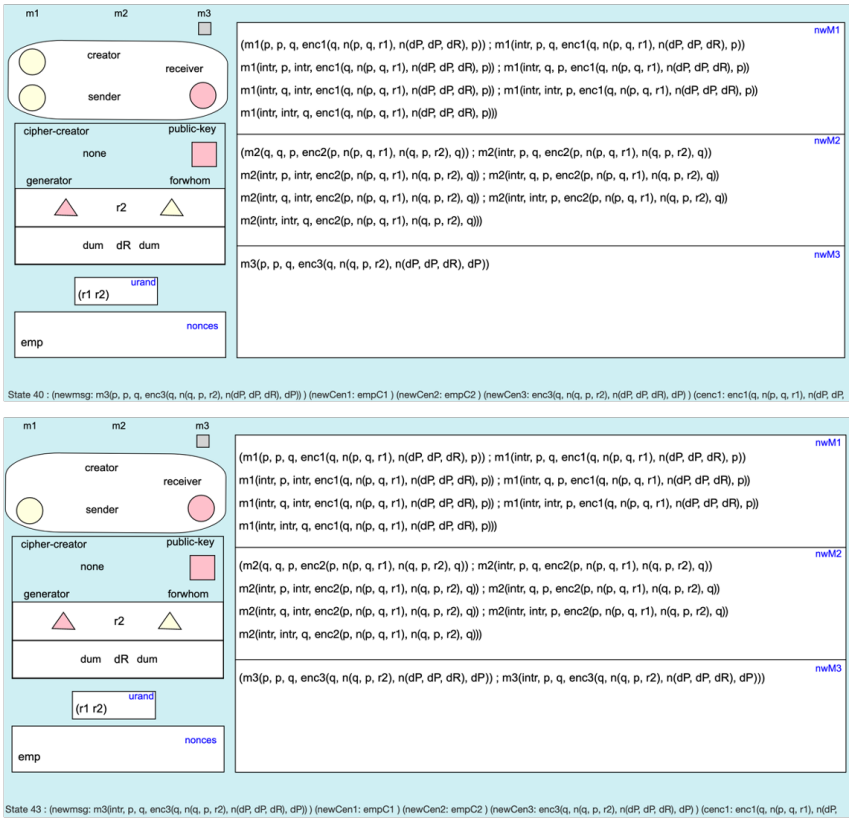


Figure 6. Some state pictures for the NSLPK protocol (3)

5. It means that the NSLPK protocol seems to enjoy the guessed characteristics.

V. Conclusion

We have graphically animated the NSLPK authentication protocol with SMGA. Observing the graphical animations based on our design allows us to guess some (non-trivial) characteristics of the state machine formalizing the NSLPK protocol. We have checked the characteristics by Maude search command. Although some model checking experiments were not completed because of the state space explosion problem, some characteristics of NSLPK have been proved [12], guaranteeing that the characteristics are invariant properties of NSLPK. One piece of our future work is to graphically animate state machines that formalize other authentication protocols, such as TLS [13], with SMGA.

References

[1] T. T. T. Nguyen and K. Ogata, "Graphical animations of state machines," in *15th DASC*, 2017, pp. 604–611.

[2] K. W. Brodlie, et al., Ed., *Scientific Visualization: Techniques and Applications*. Springer, 1992.

[3] M. T. Aung, T. T. T. Nguyen, and K. Ogata, "Guessing, model checking and theorem proving of state machine properties – a case study on Qlock," *IJSECS*, vol. 4, no. 2, pp. 1–18, 2018.

[4] T. T. T. Nguyen and K. Ogata, "Graphically perceiving characteristics of the MCS lock and model checking them," in *7th SOFL+MSVL*, 2017, pp. 3–23.

[5] D. D. Bui and K. Ogata, "Better state pictures facilitating state machine characteristic conjecture," *MTAP*, 2021.

[6] ——, "Graphical animations of the Suzuki-Kasami distributed mutual exclusion protocol," *JVLC*, vol. 2019, no. 2, pp. 105–115, 2019.

[7] G. Lowe, "An Attack on the Needham-Schroeder Public-Key Authentication Protocol," *Inf. Process. Lett.*, vol. 56, no. 3, pp. 131–133, 1995.

[8] M. Clavel, et al., Ed., *All About Maude*, ser. LNCS. Springer, 2007, vol. 4350.

[9] C. Artho, et al., "Visualization of concurrent program executions," in *31st COMPSAC*, 2007, pp. 541–546.

[10] T. T. T. Nguyen and K. Ogata, "A way to comprehend counterexamples generated by the Maude LTL model checker," in *SATE 2017*, 2017, pp. 53–62.

[11] R. M. Needham and M. D. Schroeder, "Using Encryption for Authentication in Large Networks of Computers," *Commun. ACM*, vol. 21, no. 12, pp. 993–999, 1978.

[12] K. Ogata and K. Futatsugi, "Rewriting-based verification of authentication protocols," *ENTCS*, vol. 71, pp. 208–222, 2002.

[13] T. Dierks and C. Allen, "The TLS protocol version 1.0," *RFC*, vol. 2246, pp. 1–80, 1999.